BMC Impact Event Adapters User Guide

Supporting

BMC Event and Impact Management 2.0
BMC ProactiveNet Performance Manager 8.0

November 2009

Contacting BMC Software

You can access the BMC Software website at http://www.bmc.com. From this website, you can obtain information about the company, its products, corporate offices, special events, and career opportunities.

United States and Canada

Address	BMC SOFTWARE INC 2101 CITYWEST BLVD HOUSTON TX 77042-2827 USA	**Telephone**	713 918 8800 or 800 841 2031	**Fax**	713 918 8000

Outside United States and Canada

Telephone	(01) 713 918 8800	**Fax**	(01) 713 918 8000

© Copyright 2005-2007, 2009 BMC Software, Inc.

BMC, BMC Software, and the BMC Software logo are the exclusive properties of BMC Software, Inc., are registered with the U.S. Patent and Trademark Office, and may be registered or pending registration in other countries. All other BMC trademarks, service marks, and logos may be registered or pending registration in the U.S. or in other countries. All other trademarks or registered trademarks are the property of their respective owners.

Java and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc., in the U.S. and other countries.

Perl is a trademark or registered trademark of the Perl Foundation.

UNIX is the registered trademark of The Open Group in the US and other countries.

The information included in this documentation is the proprietary and confidential information of BMC Software, Inc., its affiliates, or licensors. Your use of this information is subject to the terms and conditions of the applicable End User License agreement for the product and to the proprietary and restricted rights notices included in the product documentation.

Restricted rights legend

U.S. Government Restricted Rights to Computer Software. UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT LAWS OF THE UNITED STATES. Use, duplication, or disclosure of any data and computer software by the U.S. Government is subject to restrictions, as applicable, set forth in FAR Section 52.227-14, DFARS 252.227-7013, DFARS 252.227-7014, DFARS 252.227-7015, and DFARS 252.227-7025, as amended from time to time. Contractor / Manufacturer is BMC SOFTWARE INC, 2101 CITYWEST BLVD, HOUSTON TX 77042-2827, USA. Any contract notices should be sent to this address.

Customer support

You can obtain technical support by using the BMC Software Customer Support website or by contacting Customer Support by telephone or e-mail. To expedite your inquiry, see “Before contacting BMC.”

Support website

You can obtain technical support from BMC 24 hours a day, 7 days a week at http://www.bmc.com/support. From this website, you can

- read overviews about support services and programs that BMC offers
- find the most current information about BMC products
- search a database for issues similar to yours and possible solutions
- order or download product documentation
- download products and maintenance
- report an issue or ask a question
- subscribe to receive proactive e-mail alerts when new product notices are released
- find worldwide BMC support center locations and contact information, including e-mail addresses, fax numbers, and telephone numbers

Support by telephone or e-mail

In the United States and Canada, if you need technical support and do not have access to the web, call 800 537 1813 or send an e-mail message to customer_support@bmc.com. (In the subject line, enter **SupID**:<***yourSupportContractID***>, such as SupID:12345). Outside the United States and Canada, contact your local support center for assistance.

Before contacting BMC

Have the following information available so that Customer Support can begin working on your issue immediately:

- product information
  - product name
  - product version (release number)
  - license number and password (trial or permanent)
- operating system and environment information
  - machine type
  - operating system type, version, and service pack or other maintenance level such as PUT or PTF
  - system hardware configuration
  - serial numbers
  - related software (database, application, and communication) including type, version, and service pack or maintenance level
- sequence of events leading to the issue
- commands and options that you used
- messages received (and the time and date that you received them)
  - product error messages
  - messages from the operating system, such as `file system full`
  - messages from related software

License key and password information

If you have questions about your license key or password, contact BMC as follows:

- *(USA or Canada)* Contact the Order Services Password Team at 800 841 2031, or send an e-mail message to ContractsPasswordAdministration@bmc.com.
- *(Europe, the Middle East, and Africa)* Fax your questions to EMEA Contracts Administration at +31 20 354 8702, or send an e-mail message to password@bmc.com.
- *(Asia-Pacific)* Contact your BMC sales representative or your local BMC office.

Contents

Chapter 1 BMC Impact Event Adapters introduction 13

Adapters overview . . . 13
BMC Impact Event Adapters (BMC IEA) . . . 14
Event Adapter components . . . 15
Common characteristics of Event Adapters . . . 15
LogFile Adapter . . . 16
SNMP Adapter . . . 16
Perl EventLog Adapter for Windows . . . 17
IP Adapters . . . 18
BMC Impact Event Log Adapter for Windows . . . 18
User-defined Adapters . . . 19

Chapter 2 Configuring BMC Impact Event Adapters 21

Configuration overview . . . 22
- Adapter definitions . . . 22
- Adapter configuration tasks . . . 22

About the mcxa.conf file . . . 26
- Configuration file structure . . . 26
- Configuration file definitions . . . 28
- msend command configuration . . . 29

Parameter types . . . 29
- Global parameters . . . 30
- Common parameters . . . 32
- Specific parameters . . . 34

Applying configuration changes during run-time . . . 41
BMC Impact Event Log Adapter for Windows configuration . . . 41

Chapter 3 Defining the BMC Impact Event Adapters MAP files 43

Overview . . . 44
Default MAP files . . . 44
Custom maps and map-related files . . . 44
- Custom MAP files . . . 45

MAP file structure . . . 45
- Sections . . . 45
- Directives . . . 56

Chapter 4 Using the BMC Impact Event Adapters 59

Enabling and disabling BMC Impact Event Adapters 59
Starting and stopping the BMC Impact Event Adapters 61
Starting the engine manager process from the command line 61
Starting the Adapter processes as services. 63
Restriction for BMC Impact Event Adapters under MS Windows. 63
Stopping the BMC Impact Event Adapters 63
Using instance control to start and stop Adapters 64
Restarting an Adapter after modifying the MAP file 67
Enabling tracing 67
Trace files. 67
Trace levels 68
Errors and Adapter events 68
Adapter status events. 69

Chapter 5 Using the SNMP Adapter Configuration Manager 71

Accessing the Web-based interface 71
Publishing MIB files. 72
Viewing or editing the MAP file 74
Unpublishing MIB files 76
Installing the BAROC files in the Knowledge Base of a cell 77

Chapter 6 Using the BMC Impact Event Log Adapter for Windows 79

Overview 79
Event class definitions 79
Testing the BMC Impact Event Log Adapter for Windows 80

Chapter 7 IP Adapters 83

Overview 83
IP Adapter architecture 84
IP Adapters configuration 84
IP Adapter types 85
TCP Client Adapter 85
TCP Server Adapter 87
Telnet Adapter 90
UDP Client Adapter. 94
UDP Server Adapter 96

Chapter 8 BMC Impact Event Adapters parsers 99

What a parser does 99
LogFile, Perl EventLog for Windows, and SNMP Adapter parsers 99
IP Adapter parsers 101
Specifying a CSV parser 101
Specifying a Fixed-width (column) parser 103
Specifying Record and Field separated parser 107

Appendix A Backend processes for publishing and unpublishing MIB files 111

The publish MIB files back-end process 111
The unpublish MIB files back-end process 112

Index 115

Tables

BMC IEA versus BMC IELA comparison 18
Adapter configuration tasks 23
Parameter types 30
Global parameters 30
Common parameters 33
Perl EventLog Adapter for Windows parameters 35
LogFile Adapter parameters 36
SNMP Adapter parameters 40
Windows Event Log Adapter parameters 42
Default MAP files 44
Slot Functions 49
Operators 51
mcxa command options 61
mcxactrl.pl command options 65
IP Adapter status events 69
TCP Client Adapter configuration parameters 85
TCP Server Adapter configuration parameters 88
Telnet Adapter configuration parameters 90
UDP Client Adapter configuration parameters 94
UDP Server Adapter configuration parameters 96
CSV parser parameters 102
Fixed-Width parser parameters 104
Record and field separated parser parameters 108

Figures

mcxa.conf file structure 27
Parameter and value examples 29
Comments code example 47
Slot assignment formats 48
INIT structure 53
DEFAULT structure 53
CLASS leaf structure 54
Nested class mapping Structure 54
Class name example 55
mcsyslogd.map file excerpt 56
Map call 57
mcxactrl.pl command syntax 65
Performance Monitor in Chart mode 81
Sample TCP Client Adapter configuration 87
Sample TCP Server Adapter Configuration 89
Sample Telnet Adapter configuration 93
Sample UDP client Adapter configuration 96
Sample UDP Server Adapter configuration 98
Sample CSV parser specification example 103
Fixed-width parser specification example 105
Record and Field Separated Parser Specification Example 109

Chapter 1

BMC Impact Event Adapters introduction

This chapter provides an overview of the BMC Impact Event Adapters and their functionality.

Topics discussed in this chapter include:

Event Adapter components 15
Common characteristics of Event Adapters. 15
LogFile Adapter 16
SNMP Adapter 16
Perl EventLog Adapter for Windows 17
IP Adapters 18
BMC Impact Event Log Adapter for Windows 18

Adapters overview

A BMC Impact Manager cell collects source events from the IT assets through the use of event Adapters. Event Adapters prepare the source event data for processing by the BMC Impact Manager cell event processor.

A BMC Impact Manager cell provides the following groups of Adapters:

- BMC Impact Event Adapters collection (BMC IEA)
- BMC Impact Event Log Adapter for Windows (BMC IELA)

Each Adapter, regardless of type, consists of the following functions:

- Parser—The parser separates the data stream (source events) into records and fields using regular expressions. The BMC Impact Event Log Adapter for Windows does not use a parser.

- Map file—The map file designates how the parsed data gets mapped to event slots (fields in an event records). Map files can also be used to filter out unwanted events and to change or add data in the source event. The BMC Impact Event Log Adapter for Windows does not use a map file.

- Event class definitions—The mapped events must be translated into Basic Recorder of Objects in C (BAROC) language structures. The translated event data becomes a BMC Impact Manager cell event instance.

- Configuration—The Adapter configuration defines an instance of an Adapter type, the parser to use, parameters specific to an Adapter type, and the cell to which the Adapter should forward the events.

The Adapters run as background processes and generate self-monitoring events that can be viewed in BMC Impact Explorer.

The SNMP Adapter Configuration Manager, a component of the BMC Impact Event Adapters, automates the task of converting information from Management Information Base (MIB) files into BMC Impact Manager classes.

The SNMP Adapter Configuration Manager product requires a separate installation. See the most recent version of the *Release Notes* for information on supported platforms, prerequisites for installation, installing the product, and post-installation checks. For the current release, Java SE 6 (JDK 1.6) is supported.

BMC Impact Event Adapters (BMC IEA)

The BMC Impact Event Adapters can collect source events from

- operating system and application log files
- SNMP type 1 and type 2 traps
- message output from command line interfaces
- the Windows Event Log

The BMC Impact Event Adapters are: the LogFile Adapter, the SNMP Adapter, the Perl EventLog Adapter for Windows, and the IP Adapters.

The LogFile Adapter, the SNMP Adapter, and the Perl EventLog Adapter for Windows are preconfigured and include dedicated parsers. The IP Adapters require more extensive configuration, including the option to choose from among a number of different parsers.

The predefined Adapter instances also provide collectors for organizing events in the Events View. Administrators can modify the predefined Adapters and event classes, and can implement their own event Adapters.

Event Adapter components

Each Adapter, regardless of type, consists of the following components:

- Parser—separates the data stream (source events) into records and fields using regular expressions. The LogFile Adapter, the SNMP Adapter, and the Perl EventLog Adapter for Windows are preconfigured and include dedicated parsers. The IP Adapters require more extensive configuration, including the option to choose from among a number of different parsers. The BMC Impact Event Log Adapter for Windows does not use a parser.

- MAP file—designates how the parsed data gets mapped to event slots (fields in an event record). MAP files can also be used to filter out unwanted events and to change or add data in the source event. The BMC Impact Event Log Adapter for Windows does not use a MAP file.

- Event class definitions—The mapped events must be translated into Basic Recorder of Objects in C (BAROC) language structures. The translated event data becomes a BMC Impact Manager cell event instance.

- Configuration—defines an instance of an Adapter type, the parser to use, parameters specific to an Adapter type, and the cell to which the Adapter forwards the events.

Common characteristics of Event Adapters

Event Adapters share the following characteristics:

- All regular expressions are managed by the Perl interpreter, and they must be in Perl 5 style.

- Several instances of the same Adapter type can run at the same time. For example, two SNMP Adapters can be configured to listen on two different ports. Similarly, several LogFile Adapters can monitor several log files with completely different settings.

- Each instance of an Adapter is related to a MAP file. The MAP file is a text file that manages the translation between a specific event coming from an external source and a BMC Impact Manager event. It consists of a set of statements, conditions, and assignments.

- All BMC Impact Event Adapters store their configurations in the same configuration file, **mcxa.conf**. You can specify a different configuration file, if necessary.

- The **mcxa** process for the BMC Impact Event Adapters can run as a daemon on UNIX systems and as a service on Windows systems.

LogFile Adapter

The LogFile Adapter is a file reader that can be used with any text file containing records that can be recognized by Perl regular expressions that describe the record and the record separator.

Although the LogFile Adapter is meant to be a generic Adapter for any text-based log file, special Adapter configurations and MAP files are supplied to monitor the most common UNIX log file produced by the syslog process and Apache log files. The **mcxa.conf** file includes sample Adapter configurations for each of these different functions.

NOTE

On Windows systems, the LogFile Adapter detects that the log file has changed if its size has decreased. This means that the LogFile Adapter will not function properly if the log file is replaced by a larger file. If such a situation occurs, delete the older log file about 30 seconds before replacing it. Another solution is to modify the last modification time of the configuration file.

SNMP Adapter

The SNMP (Trap) Adapter consists of a UDP SNMP server listening for SNMP traps. It includes a tool to convert information from Management Information Base (MIB) files into BMC Impact Manager classes and other data used to format traps into BMC Impact Manager events.

The SNMP Configuration Manager, a component of the BMC Impact Event Adapters, automates the tasks for converting information from MIB files into BMC Impact Manager classes.

The SNMP Configuration Manager has a web-based interface that enables you to perform the following tasks:

- publish MIB files

  This process converts information from MIB files into BMC Impact Manager classes. The SNMP Configuration Manager uses the existing **mib2map.pl** utility to automate this process.

- view and edit the MAP file

  The MAP file manages the translation between a specific event coming from an external source and a cell event. Each instance of an Adapter is related to a MAP file. To get the name of the MAP file that has to be configured, the SNMP Configuration Manager refers to the `MapFile` parameter in the first enabled instance of the SNMP Adapter in the **mcxa.conf** file. The **mcxa.conf** file is located in the **MCELL_HOME\etc** directory. MCELL_HOME is the path to the cell and BMC Impact Event Adapters configuration and executable files.

  If you have not specified a file name for the `MapFile` parameter, the SNMP Configuration Manager uses the **mcsnmptrapd.map** file name by default. This book uses **mcsnmptrapd.map** to refer to the file that is configured for the `MapFile` parameter.

- unpublish MIB files

  The process of unpublishing MIB files removes information about the selected MIB files from the **mcsnmptrapd.map** file.

Before you install the SNMP Configuration Manager, you must install the BMC Impact Event Adapters and enable the SNMP Adapter.

Perl EventLog Adapter for Windows

The Perl EventLog Adapter for Windows is a Windows-only Adapter written in Perl that runs in the **mcxa** process. It monitors the system, security, and application events generated by a Windows operating system, translates the events, and forwards the events to a cell. This Adapter is obsolete and is supplied for backward-compatibility only.

BMC Software recommends that you use the BMC Impact Event Log Adapter for Windows instead to monitor the Microsoft Windows Event Log. The BMC Impact Event Log Adapter for Windows is installed separately. It uses a binary application and runs as a separate service. For more information, see “BMC Impact Event Log Adapter for Windows” on page 18.

IP Adapters

The IP Adapters use the various protocols of the IP protocol suite to establish connections with programs from which you may want to collect BMC Impact Manager event data. The IP Adapters are as follows:

- TCP Client Adapter
- TCP Server Adapter
- Telnet Adapter
- UDP Client Adapter
- UDP Server Adapter

For more information, see Chapter 7, "IP Adapters."

BMC Impact Event Log Adapter for Windows

The BMC Impact Event Log Adapter for Windows (BMC IELA) monitors the system, security, and application events generated by a Windows operating system, translates the events, and forwards the events to a cell. The BMC Impact Event Log Adapter for Windows provides predefined parsing, configuration, event classes, and collectors for organizing events in the Events View.

The BMC Impact Event Log Adapter for Windows offers the same functionality as the Perl EventLog Adapter for Windows without requiring you to install Perl on your computer. The implementation of the BMC Impact Event Log Adapter for Windows differs from that of the other BMC Impact Event Adapters. Table 1 on page 18 describes these differences.

Table 1 BMC IEA versus BMC IELA comparison

Criterion	BMC Impact Event Adapters (IEA)	BMC Impact Event Log Adapter for Windows (IELA)
Adapters included in package	■ Perl LogFile Adapter for Windows ■ LogFile Adapter ■ SNMP Adapter ■ IP Adapters	BMC Impact Event Log Adapter for Windows
Process	**mcxa.pl** process (Engine Manager)	binary executable
Platform	Windows and UNIX	Windows
Windows Service name	BMC Impact Event Adapters	BMC Impact Event Log Adapter for Windows
Configuration location	**mcxa.conf** file	Windows registry

For more information about the BMC Impact Event Log Adapter for Windows, see Chapter 6, “Using the BMC Impact Event Log Adapter for Windows.”

User-defined Adapters

Administrators can modify existing Adapters or create and implement their own Adapters. User-defined Adapters can be easily integrated into the system in the same manner as the Perl EventLog Adapter for Windows, the LogFile Adapter, and the SNMP Adapter. User-defined Adapters must be Perl packages that inherit from the `MA::Engine` module and define some functions that collect the desired external data. Use of such an Adapter requires only that its name be entered as the `Engine` parameter in the configuration file after which the Engine Manager loads it dynamically.

Chapter 2

Configuring BMC Impact Event Adapters

This chapter describes the function and contents of the **mcxa.conf** file, emphasizing the structure and contents of the configuration specifications for the various Adapters.

NOTE

The BMC Impact Event Log Adapter for Windows runs independently of the mcxa process and the **mcxa.conf** file.

Configuration overview 22
- Adapter definitions 22
- Adapter configuration tasks 22

About the mcxa.conf file 26
- Configuration file structure 26
- Configuration file definitions 28
- msend command configuration 29

Parameter types 29
- Global parameters 30
- Common parameters. 32
- Specific parameters 34

Applying configuration changes during run-time 41
BMC Impact Event Log Adapter for Windows configuration 41

Configuration overview

You configure the BMC Impact Event Adapters by modifying the parameters that control their behavior. All parameters are recorded in the **mcxa.conf** file.

- *Global* parameters control certain behavioral aspects of all Adapters running on the engine manager (**mcxa** process). Global parameters are listed in the Global Parameters section of the **mcxa.conf** file.
- *Common* parameters behave like global parameters. Common parameters are listed in the Common Parameters section of the **mcxa.conf** file.
- *Specific* parameters dictate selected aspects of the behavior on an individual Adapter instance. These parameters are recorded in the **mcxa.conf** file in a definition that is specific to an Adapter instance.

Adapter definitions

The **mcxa.conf** file contains definitions for all of the Adapters running on the computer, as follows:

- All types of BMC Impact Event Adapters are listed in and are run from the **mcxa.conf** file. For example, IP Adapters do not use a separate **mcxa.conf** file from LogFile Adapters.
- Multiple Adapter instances can be defined in the file. For example, you can have three different TCP Client Adapters running, each with a unique configuration.

For more information about the structure of the **mcxa.conf** file, see "About the mcxa.conf file" on page 26.

Adapter configuration tasks

To configure the global aspects of the Adapters environment and the parameters specific to each Adapter instance, perform the tasks in the order described in Table 2 on page 23.

NOTE

You can add, remove, or modify global or common parameters, Adapter instance definitions, or parser definitions in the **mcxa.conf** file at any time.

If you make a change when the engine manager (the **mcxa** process) is stopped, your changes will be applied when you restart the engine manager. You can also make changes during run-time. To ensure that these changes are recognized, see "Applying configuration changes during run-time" on page 41.

Table 2 Adapter configuration tasks (part 1 of 4)

Phase	Action	Details	Where to go
Accessing the configuration file	Open the **mcxa.conf** file.	The file is located at ***MCELL_HOME*\etc\mcxa.conf**	"About the mcxa.conf file" on page 26
Configuring the Adapter environment	Specify values for global parameters.	Some parameters are required and some are optional.	"Global parameters" on page 30
	Specify values for common parameters.	Some parameters are required and some are optional.	"Common parameters" on page 32
Configuring Adapters	Decide which Adapters you want to use.	Determine what you want to accomplish with the Adapters and select the Adapters appropriate to the task.	"IP Adapter types" on page 85

Table 2 Adapter configuration tasks (part 2 of 4)

Phase	Action	Details	Where to go
Configuring Adapters, *continued*	Gather the information required to define the Adapter instances.	Review the specific parameters associated with each Adapter that you want to use. Make a list of values that you want to add or modify for each Adapter instance (such as port numbers). You must specify which parser each of your IP Adapter instances will use. Review the types of parsers available. BMC Software recommends that you create a brief planning sheet for each Adapter, listing the information that you want to include in the Adapter definition.	■ "Perl EventLog Adapter for Windows parameters" on page 34 ■ "LogFile Adapter parameters" on page 35 ■ "SNMP Adapter parameters" on page 40 ■ "IP Adapter types" on page 85 ■ "IP Adapter parsers" on page 101
	Modify or add Adapter definitions.	The **mcxa.conf** file contains sample Adapter definitions for all supported parameters. 1. Copy these definitions below the **Write your configuration below** heading at the end of the **mcxa.conf** file. Make a copy for each Adapter instance that you require. 2. Add or modify Adapter parameters as required for each Adapter instance.	not applicable

Table 2 Adapter configuration tasks (part 3 of 4)

Phase	Action	Details	Where to go
Configuring Adapters, *continued*	Define any required IP Adapter parsers.	The **mcxa.conf** file contains sample parser definitions for all supported parameters. 1. Copy these definitions below the **Write your configuration below** heading at the end of the **mcxa.conf** file. Make a copy for each parser instance that you require. 2. Add or modify parser parameters as required for each unique parser instance.	"IP Adapter parsers" on page 101
	(*optional*) Enable Adapters.	You can enable Adapters at any time after they have been configured. However, some Adapters may require additional preparatory work before they are enabled: ■ If you intend to use a custom mapping, review Chapter 3, "Defining the BMC Impact Event Adapters MAP files," on page 43 and do any required preparatory work before enabling any Adapters using custom mappings. ■ Before enabling an SNMP Adapter, perform any required configuration involving the **mib2map** utility. **Note**: When enabling an IP Adapter, you must also enable the parser that it uses (if that parser is not already enabled).	■ "Custom maps and map-related files" on page 44 ■ *Installation Guide*
	Save and close the **mcxa.conf** file.	not applicable	not applicable

Table 2 Adapter configuration tasks (part 4 of 4)

Phase	Action	Details	Where to go
Configuring the SNMP **mib2map** utility	Configure the **mib2map** utility (SNMP Adapter only).	The **mib2map** utility specifies how SNMP data is converted to cell events. You must perform some configuration of this utility to ensure that SNMP data is presented correctly in the events. **Note**: Perform this task before you enable any SNMP Adapter instances.	
Enabling Adapters	Remove or comment out **DISABLE** in the Adapter definitions.	Removing or commenting out **DISABLE** in an Adapter definition enables it immediately. Do this only when you are ready to collect data. **Notes**: ■ If you did not enable the Adapters when you first defined them, then you must re-open the **mcxa.conf** file, remove **DISABLE** from the required definition, and save and close the file. ■ When enabling an IP Adapter, you must also enable the parser that it uses (if that parser is not already enabled).	*"Enabling and disabling BMC Impact Event Adapters" on page 59*

About the mcxa.conf file

The Adapters are configured in a single configuration file, ***MCELL_HOME*\etc\mcxa.conf**, by default. You can use other configuration files by starting the Perl process with the CLI `-c` option.

Configuration file structure

Figure 1 on page 27 shows the structure of the default Adapters configuration file, **mcxa.conf**.

Figure 1 mcxa.conf file structure

```
# Unnamed section is considered as [default]
# [default] is optional
<global_parameter1> = <gval1>
<global_parameter2> = <gval2>
# ...
[1st Adapter]
<ada1_param1> = <a1val1>
<ada1_param2> = <a1val2>
# ...
[2nd Adapter]
<ada1_param1> = <a1val1>
<ada1_param2> = <a1val2>
# ...
```

Generally, global parameters either apply to the Adapter engine or serve as default parameters for all specific Adapters. Specific parameters for the specific Adapters are located after the `[ . . . ]` Adapter sections.

The Adapters are specified by the `Engine` parameter:

```
Engine = <engine_module>
```

`<engine_module>` is the name of the Perl module that is loaded by the Engine Manager Perl module. It must be located in the standard Perl library directory or in a subdirectory of ***MCELL_HOME*\lib\perl**. It usually can be user defined. The predefined default engines are as follows:

- `Engine = MA::ELogfile` for any log file management, including UNIX syslog
- `Engine = MA::ESnmpTrap` for SNMP trap management
- `Engine = MA::EEventLog` for Perl EventLog for Windows management

The IP Adapter definitions vary slightly, with the inclusion of `Adapter::` in the path.

- `Engine = MA::Adapter::TcpClient`
- `Engine = MA::Adapter::TcpServer`
- `Engine = MA::Adapter::Telnet`
- `Engine = MA::Adapter::UdpClient`
- `Engine = MA::Adapter::UdpServer`

You can configure several identical Adapters in **mcxa.conf** to monitor several log files or to listen from several SNMP ports, for example.

The name of the destination cell is provided in the common parameter, `ServerName`. It can be identified in the specific section, so that two different Adapters can send events to different cells. In addition, `ServerName` can contain the name of several cells, separated by colons. In this case the Adapter tries to send events to one the cells in the list, always trying the first cell (the primary cell) in the list when any other cell goes down.

Other information about the configuration, the parameters list, and examples are located in the header of the **mcxa.conf** file.

Configuration file definitions

The **mcxa.conf** file contains comments, sections, and parameter definitions.

Comments

Anything that follows the **#** character on a line is ignored, including the **#** character, unless it is included inside single or double quotes.

WARNING

BMC Software recommends that you do not put comments on lines that contain quoted values. In ambiguous cases, quotes take precedence, which may cause unintended results.

Sections

The configuration file is divided into sections beginning with the default section at the beginning of the file. Each subsequent section represents an instance of a specific Adapter. An Adapter section begins with a section name in square brackets "`[]`" and ends at the beginning of another section or at the end of the file.

Configuration file parameters

The configuration file primarily contains parameter definitions in the form of

```
<param> = <value>
```

The <param> and <value> variables are defined as follows:

Variable	Description
<param>	The parameter name. It cannot contain blank spaces.
<value>	The parameter value. The first character of <value> is the first character of the line that is not blank and not a pound sign (#). <value> can be enclosed within double or single quotes.

Figure 2 shows examples of parameters and values. Section headers that indicate the type of entity being described with the parameters that follow are displayed in **magenta** text. Parameter names are displayed in **red** text. Parameter values are displayed in **blue** text.

Figure 2 Parameter and value examples

```
[TcpClient]
Engine = MA::Adapter::TCPClient
Parser = MySeparatorParser
MapFile = mctcpclt.map
RemoteHost = localhost
RemotePort = 13       # date/time service

[MyTcpServer]
Engine = MA::Adapter::TcpServer
MaxClients = 10
RestartInterval = 30
MapFile = mctcpsrv.map
LocalHost = 127.0.0.5
LocalPort = 1999
Parser = MyFixedWidthParser
```

msend command configuration

Events produced by the Adapters are sent to the cell with the BMC Impact Manager msend command. To set up non-default parameters, configure msend with ***MCELL_HOME*\etc\msend.conf** or ***MCELL_HOME*\etc\mclient.conf**.

Parameter types

Table 3 on page 30 describes the parameters types available for use with the BMC Impact Event Adapters.

Table 3 Parameter types

Parameter type	Description
Global	Global parameters influence the global Perl process and must be defined in the default section of the configuration file.An example of a global parameter is `ReadPerEngine`.
Common	Common parameters are common to all Adapters, but unlike global parameters, common parameters can be different for different Adapter instances. An example of a common parameter is `MapFile`. These parameters are defined in the default section of the **mcxa.conf** file, in the specific Adapter sections, or both. If a common parameter is defined in the default section, it applies to all Adapters unless it is also defined in a specific Adapter section. If it is defined in a specific section, it applies only to that Adapter.
Specific	Specific parameters apply only to one specific type of Adapter. They are generally defined in specific Adapter sections; however, by using the `Engine` parameter, you can define default specific parameters in the default section of the **mcxa.conf** file that apply to all relevant specific Adapters.

Global parameters

Table 4 describes the global parameters that can be used with the Adapters and each parameter's characteristics.

Table 4 Global parameters (part 1 of 3)

Parameter	Type	Unit	Default value	Description
`EventsPerPoll`	Integer > 0	events	5	used only if `ReadsPerEngine = 1`; defines the maximum number of events each Adapter generates per poll cycle. If `ReadsPerEngine = 1`, records are read until the number of events created equals the value of the `EventsPerPoll` parameter. Otherwise, the maximum reads per engine per poll cycle is set by the value of the `ReadsPerEngine` parameter.
`PollInterval`	Integer $\geq$ 0	sec	5	sleep interval during which all Adapters are idle
`ReadsPerEngine`	Integer > 0	reads	5	defines the maximum number of reads each Adapter performs per poll cycle. If `ReadsPerEngine = 1`, records are read until the number of events created equals the value of the `EventsPerPoll` parameter. Otherwise, `ReadsPerEngine` equals the maximum reads per engine per poll cycle.

Table 4 Global parameters (part 2 of 3)

Parameter	Type	Unit	Default value	Description
TraceFile	File name	none	**mcxa.trace**	name of the trace file If the name does not contain a path, the trace file is saved to the default trace file directory, ***MCELL_HOME*\tmp\Adapters**.
TraceFileCount	Integer ≥ 0	none	10	maximum number of renamed trace files that are kept in the trace file directory When the number of files exceeds the specified limit, the oldest trace file is deleted. To expand the amount of saved trace information, increase the value of *one* of the following parameters: ■ TraceSizeMax (default = 1,000,000) ■ TraceFileCount (default = 10)
TraceLevel	Integer (0–6)	none	3	trace level Valid values: ■ **0** – disables all traces ■ **1** – header messages ■ **2** – fatal messages ■ **3** – major errors ■ **4** – minor errors ■ **5** – normal output ■ **6** – verbose output **Note**: TraceLevel = 6 produces very large trace files and is recommended only for debugging.

Table 4 Global parameters (part 3 of 3)

Parameter	Type	Unit	Default value	Description
TraceSizeMax	Integer >= 0	byte	1,000,000	maximum trace file size When the Adapter detects that the maximum size is reached, the trace file is closed and renamed **mcxa0.trace**. **Notes**: ■ Old trace files rotate as 0->1, ..., n->n+1. ■ The number of trace files to keep is managed with the TraceFileCount parameter.
TraceSizePeriod	Integer > 0	none	10	frequency, in number of messages logged, at which the Adapter checks the size of the trace file A trace file is archived when it exceeds the size specified with the TraceSizeMax parameter.

Common parameters

Table 5 on page 33 describes the common parameters that can be used with the Adapters.

Table 5 Common parameters (part 1 of 2)

Parameter	Type	Description
`Engine`	Perl module name	name of the Adapter This parameter is required. Default values are as follows: ■ `Engine = MA::ELogfile` for any log file management, including UNIX syslog ■ `Engine = MA::ESnmpTrap` for SNMP trap management ■ `Engine = MA::EEventLog` for Perl EventLog for Windows management The IP Adapter definitions vary slightly, with the inclusion of `Adapter::` in the path: ■ `Engine = MA::Adapter::TcpClient` ■ `Engine = MA::Adapter::TcpServer` ■ `Engine = MA::Adapter::Telnet` ■ `Engine = MA::Adapter::UdpClient` ■ `Engine = MA::Adapter::UdpServer`
`MapFile`	File name	map file name The map file *must* be located in the **MCELL_HOME\etc** directory. Full paths are *not* accepted. Default map name values depend on the Adapter type. Default map names are listed in "Default MAP files" on page 44.
`MapFunctionsName`	File name	name of a Perl script or Perl module that contains user-defined functions to be used inside the **.map** file The use of this parameter is now deprecated. Use the `REQUIRE` directive, described in "REQUIRE directive" on page 56, from inside the **.map** file.

Table 5 Common parameters (part 2 of 2)

Parameter	Type	Description
MapUseDefaultClass	Bool (0-1)	indicates whether incoming events that do not match a CLASS described in the map file are mapped to the DEFAULT class Valid values: ■ **0** – events are dropped (this is the default). ■ **1** – events are mapped to the DEFAULT class.
ServerName	Cell name	name of the cell to which events processed by the Adapter are sent The cell parameters must be found in *MCELL_HOME***etc****mcell.dir.** ServerName can contain the names of several cells, separated by colons, such as Cell1:Cell2:Cell3:. . . . Events are sent to these cells according to their numerical order. For more information, see the mposter/msend documentation inthe *Administration Guide*. If ServerName is not defined, events are formatted in BAROC and sent to the standard output.

Specific parameters

Each Adapter definition includes specific parameters that are used to configure only the Adapter instance in which they are listed.

Each type of Adapter has a different set of specific parameters.

- For Perl EventLog Adapter for Windows parameters, see "Perl EventLog Adapter for Windows parameters" on page 34.
- For EventLog Adapter parameters, see "LogFile Adapter parameters" on page 35.
- For SNMP Trap Adapter parameters, see "SNMP Adapter parameters" on page 40.
- For IP Adapter parameters, see Chapter 7, "IP Adapters," on page 83.

Perl EventLog Adapter for Windows parameters

This section describes the specific parameters used by the Perl EventLog Adapter for Windows. For details about configuring the BMC Impact Event Log Adapter for Windows, see "BMC Impact Event Log Adapter for Windows configuration" on page 41.

To use the Perl EventLog Adapter for Windows, set the Engine parameter in the **mcxa.conf** file as follows:

```
Engine = MA::EEventLog
```

The default map file is **mceventlog.map**.

Table 6 describes the Perl EventLog Adapter parameters.

Table 6 Perl EventLog Adapter for Windows parameters

Parameter	Description
ELResendAll	enables restarting the reading of the EventLog If this parameter is set to 1, all the events in the eventlog are resent one time. Default: **0** **WARNING**: The BMC Impact Event Log Adapter for Windows EResendAll parameter is equivalent to the Perl ELResendAll parameter. The parameter names are similar but the values are different. For this reason, BMC Software recommends that you exercise caution when using them. Be sure to use a value of 0 or 1 for the Perl EventLog Adapter ELResendAll parameter and a value of Yes or No for the BMC Impact Event Log Adapter EResendAll parameter. For details about configuration the parameters for the BMC Impact Event Log Adapter for Windows, see "BMC Impact Event Log Adapter for Windows configuration" on page 41.
ELRecoveryIntervalMin	recovery time interval, specified in minutes Unless ELResendAll is set to 1, an Adapter processes events that arrived in the log since it was stopped. ELRecoveryIntervalMin prevents processing of old events if the Adapter has been stopped for some time. That is, the Adapter will process the events that are younger than ELRecoveryIntervalMin. Default: 1440 minutes (1 day)
ELLogNotRead	a string that contains the names, separated by a colons, of the event logs the user does not want to read, such as System, Application, or Security

LogFile Adapter parameters

To use the LogFile Adapter for Windows, set the Engine parameter in the **mcxa.conf** file as follows:

```
Engine = MA::ELogfile
```

The default map file is **mclogfile.map**.

Table 7 on page 36 describes the LogFile Adapter parameters.

Table 7 LogFile Adapter parameters (part 1 of 4)

Parameter	Type	Unit	Default value	Description
LogFile	File name	none	standard input	full path to the monitored log file LogFile supports a single file name. Use several Adapter instances to monitor several log files.
LogFieldSeparator	Regex	none	\s+	a regular expression, often a character such as " , ", that delimits the attributes of an event when LogRegExpr is empty **Note**: No syntactical analysis of the event occurs with this parameter, such as detecting a LogFieldSeparator inside quoted strings.
LogFlushPosPeriod	Integer ≥ 0	none	10	number of events after which the log position is saved to disk The position file is flushed after the number of incoming log entries reaches the number specified with LogFlushPosPeriod. **Note**: Setting LogFlushPosPeriod to a low value in an environment where many short events arrive at a high rate can degrade performance. See the LogRememberPos parameter on page 38 for more information.
LogKeepEmpty	Bool (0-1)	none	0	indicates whether to keep or drop empty events or events only constituted by blanks Valid values: ■ **0** – drop events ■ **1** – keep events
LogMaxCount	Integer ≥ 0	none	10	maximum number of log files when log rotation is enabled For example, if LogMaxCount=5, <logfile>0–4 are saved. <logfile>i is always newer than <logfile>i+1. If LogMaxCount=-1, all log files are kept. **Note**: Specify **0** if you do not want to keep log files.
LogMaxSize	Integer > 0	byte	1,000,000	log file maximum size, in bytes When the LogMaxSize value is reached, the log file is rotated. For more information, see the LogRotate parameter on page 39. If LogRotate=0, LogMaxSize is not used.

Table 7 LogFile Adapter parameters (part 2 of 4)

Parameter	Type	Unit	Default value	Description
LogProcessName	String	none	empty	(UNIX only) name of the process receiving the kill HUP command LogProcessName is not used if LogSupportKillHUP=0. See LogSupportKillHUP on page 39.
LogReadAll	Bool (0-1)	none	0	indicates whether the whole log file is read the first time the Adapter starts Valid values: ■ **0** – is not read ■ **1** – is read
LogReadAllReopen	Bool (0-1)	none	1	indicates whether an Adapter that detects that a log file has changed opens a new log file from the beginning or the end Valid values: **0** – opens from the end **1** – opens from the beginning
LogRecordSeparator	Regex	none	\n	regular expression, often a simple character or string, that delimits two events **Note**: If LogRecordSeparator is empty (LogRecordSeparator=“”), an event is found as soon as the regular expression encounters a match. The matching regular expression is consumed and the remainder of the file is used to detect the next event.

Table 7 LogFile Adapter parameters (part 3 of 4)

Parameter	Type	Unit	Default value	Description
LogRegExpr	Regex	none	empty	contains a regular expression, in Perl 5 style, used to match an event The pattern variables $1, $2, ... are saved in $<logVarPrefix>1, $<logvarPrefix>2, ..., and are declared in the INPUT_VARIABLES section of the **.map** file. $<logVarPrefix>0 contains the whole pattern. $complete contains the whole string between two LogRecordSeparator parameters. If the regular expression matches the whole string between two LogRecordSeparator parameters, both $complete and $<logVarprefix>0, contain the whole event. If LogRegExpr is empty (logRegExpr=""), LogFieldSeparator is used instead.
LogRegExprGlobal	Bool (0-1)	none	1	indicates whether matching with the LogRegExpr parameter is performed globally Matching is global when you use the /g modifier of a Perl regular expression. In this case, the pattern is applied as many times as possible to the string. Each time the pattern matches, it is stored in $<logvarPrefix><n> variables, as explained in LogRegExpr on page 38. For example, if LogRegExpr="\w+" and LogRegExprGlobal=1, every word of the event is stored in $<logvarPrefix><n> variables (n>=1).
LogRememberPos	Bool (0-1)	none	0	indicates whether the Adapter persistently remembers the position of the last log entry Valid values: ■ **0** – does not remember position ■ **1** – remembers position **Note**: This information is stored in **MCELL_HOME\log\Adapters\perl\elogfile.pos**.

Table 7 LogFile Adapter parameters (part 4 of 4)

Parameter	Type	Unit	Default value	Description
LogRotate	Bool (0-1)	none	0	indicates whether a new logfile is created when the size of the logfile becomes greater than the value of the LogMaxSize parameter If a new logfile is created, the old logfile is renamed or deleted, depending on the value of LogMaxCount. If the old log file is kept, it becomes <logfile>0[.<ext>], <logfile>0[.<ext>] becomes <logfile>1[.<ext>], ... and <logfile><logMaxCount>[.<ext>] is deleted if the value of LogMaxCount is different from -1.
LogSmartOpen	Bool (0-1)	none	1	indicates whether the part of the log file that has not been consumed is considered as being appended to the beginning of the newer log file, when the log file is replaced Valid values: ■ **0** – not appended ■ **1** – appended
LogStatPeriod	Integer ≥ 5	sec	15	indicates whether the Adapter regularly performs the stat command to check whether the logfile exists, to check its i-node on UNIX platforms, or to check whether its size has not decreased on Windows platforms
LogSupportKillHUP	Bool (0–1)	none	0	(UNIX only) If LogSupportKillHUP=1, a kill -HUP command is launched on LogProcessName at each rotation of the log file. Note that if LogSupportKillHUP=0 and LogRotate=1, it is important to ensure that the process that generates the events can detect the rotation.
LogVarPrefix	String	none	"varlog"	prefix of the input variables that corresponds to the matching of the event variables $<logVarPrefix>0 to $<logVarPrevix><n>, where <n> is the maximum number of matches. The prefix must be defined in the INPUT_VARIABLES section of the **.map** file. Alternatively, the notation $<logVarPrefix>0-i can be used to match any arbitrarily long list of patterns.

SNMP Adapter parameters

To use the SNMP Adapter, set the `Engine` parameter in the **mcxa.conf** file as follows:

```
Engine = MA::ESnmpTrap
```

The default map file is **mcsnmptrapd.map**.

Table 8 describes the SNMP Adapter parameters.

Table 8 SNMP Adapter parameters

Parameter	Type	Description
`SnmpDatFile`	File name	name of the **.dat** file that contains information used to translate incoming traps If the parameter value is a relative path, the file must be located in the ***MCELL_HOME*\etc** directory. The **.dat** file is an enhanced version of the old **.oid** file. It can contain additional information to map enumerations and to extract indexes. This file contains the results of the output of the cell's **mib2map** tool. Do not attempt to create this file manually. Default: **mcsnmptrapd.dat**
`SnmpGetIndexes`	Bool (`0`–`1`)	starts and stops index extraction, mainly for performance purposes By default, the indexes, such as the suffixed dotted numbers of the object identifiers (OIDs), are extracted and stored in the special `$idx` input variable. `$idx` is a reference on a list of lists. Valid values: ■ **0** – starts ■ **1** – stops
`SnmpLocalAddr`	IP address	specifies which interface to use on a computer with two or more interface cards
`SnmpOIDFile`	File name	name of the file containing translations from SNMP OIDs to strings If the parameter value is a relative path, the file must be located in the ***MCELL_HOME*\etc** directory. The use of `SnmpOIDFile` is now obsolete. Use the complete `SnmpDatFile` parameter instead. Default: **mcsnmptrapd.oid**

Table 8 SNMP Adapter parameters

Parameter	Type	Description
SnmpPort	Integer > 0	port of the UDP SNMP server Default: **162**
SnmpTrapLength	Integer > 0	initial value of the buffer that receives SNMP traps Default: **8192** Use the SnmpTrapLength parameter default setting. If you must modify it, be aware that an MC_ADAPTER_ERROR will occur if the SnmpTrapLength value is smaller than the actual size of the trap.

NOTE

The BMC Impact Manager SNMP Adapter does not respond to INFORM messages that it may receive, but it does parse the SNMP message and generate an event.

Applying configuration changes during run-time

You can modify the **mcxa.conf** configuration file during run-time. When the engine manager detects that the configuration file has changed, it stops the currently running affected Adapters, and then restarts them according to the new values in the configuration file.

BMC Impact Event Log Adapter for Windows configuration

The configuration parameters for the BMC Impact Event Log Adapter for Windows are stored in the following Windows registry key:

```
"HKEY_LOCAL_MACHINE\SOFTWARE\BMC Software\Eventlog Adapter"
```

NOTE

For Microsoft Windows 2003 Server X64 (32-bit compatibility mode) only, the correct path to the registry key for the specific parameters for the BMC Impact Event Log Adapter for Windows is "HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\BMC Software\Eventlog Adapter".

You can configure the BMC Impact Event Log Adapter for Windows Adapter by modifying the specified parameter values in the Windows registry. Table 9 describes the configuration parameters.

Table 9 Windows Event Log Adapter parameters

Parameter	Description
EResendAll	enables restarting the reading of the EventLog If this parameter is set to Yes, all the events in the eventlog are resent one time. Default: No **WARNING**: The BMC Impact Event Log Adapter for Windows EResendAll parameter is equivalent to the Perl ELResendAll parameter. The parameter names are similar but the values are different. For this reason, BMC Software recommends that you exercise caution when using them. Be sure to use a value of 0 or 1 for the Perl EventLog Adapter ELResendAll parameter and a value of Yes or No for the BMC Impact Event Log Adapter EResendAll parameter.
ERecoveryIntervalMin	recovery time interval, specified in minutes Unless EResendAll is set to Yes, an Adapter processes events that arrived in the log since it was stopped. ELRecoveryIntervalMin prevents processing of old events if the Adapter has been stopped for some time. That is, the Adapter will process the events that are younger than ELRecoveryIntervalMin. Default: 1440 minutes (1 day)
PollInterval	time interval, in seconds, at which the Adapter checks for new log entries
ELogNotRead	A string that contains the names, separated by a colon ':', of the event logs the user does not want to read, such as System, Application, or Security.

Chapter 3

Defining the BMC Impact Event Adapters MAP files

This appendix discusses how to define the syntax and use a MAP file.

Default MAP files 44
Custom maps and map-related files 44
    Custom MAP files 45
MAP file structure 45
    Sections 45
    Directives 56

Overview

When an event is read by an Adapter, the Adapter must convert the event from its internal representation, such as a Perl hash, into the cell-compatible format, the BAROC language. The structure of the conversion is controlled by a special configuration file, the MAP file. Each Adapter uses its own MAP file to produce cell events.

BAROC classes for Adapters are defined in the **mcxa.baroc** file, which is installed in the default Knowledge Base of each cell Event Processor. The Adapters create a copy of this file in the **MCELL_HOME\etc** directory. This file also contains classes for standard events sent by `msend` and `mposter`.

Default MAP files

Table 10 lists the default names for the MAP files of the different BMC Impact Event Adapters.

- On UNIX platforms, MAP files are located in the *MCELL_HOME*/**etc** directory.
- On Windows platforms, MAP files are located in the *MCELL_HOME***etc** directory.

Table 10 Default MAP files

Adapter	Default MAP file name
Perl EventLog for Windows	**mceventlog.map**
LogFile	**mclogfile.map**
SNMP Trap	**mcsnmptrapd.map**
Apache LogFile	**mcapache.map**
UNIX syslog	**mcsyslogd.map**
TCP Client	**mctcpclt.map**
TCP Server	**mctcpsrv.map**
Telnet	**mctelnet.map**
UDP Client	**mcudpclt.map**
UDP Server	**mcudpsrv.map**

Custom maps and map-related files

If you want to expand map functionality beyond that offered using the default maps, you can create custom map files that produce the event formats that you require.

Custom MAP files

To use a MAP file other than the default, specify the path and name of the map file as the value of the `MapFile` parameter in the Adapter specification in the **mcxa.conf** file. For example:

```
MapFile = mymap.map
```

MAP file structure

The structure of a MAP file is line-oriented. Each line can be one of the six (6) types of recognized statements:

- a comment
- a define statement
- an input variables statement
- an initialization statement
- a default class statement
- a class statement

Sections

The MAP file can be divided into the following sections:

- `INPUT_VARIABLES`

  In this section, input variables from the Adapter are defined. Input variables statements belong in this section. See "INPUT_VARIABLES section" on page 46.

- `INIT`

  In this section common initialization of the slots is performed. `Init` statements belong in this section. See "INIT section" on page 52.

- `DEFAULT`

  A default class is defined in this section. Default class statements belong in this section. See "DEFAULT section" on page 53.

- CLASS

  The mapping between BMC Impact Manager classes and internal Adapter classes is defined in the CLASS section. Class statements belong in this section. See "CLASS section" on page 53.

INPUT_VARIABLES section

Input variables take their value from the parsing of data received by the Adapter. Parsers do this by parsing a log file from a Syslog or LogFile Adapter, decoding a packet from an SNMP Adapter, or using a dedicated API from a user-defined Adapter. Variables are stored internally by the Adapter in the form a Perl Hash table. Variables must be declared inside the INPUT_VARIABLES structure in order to be recognized and used by the MAP file. The input variables are not chosen arbitrarily. They must correspond to the internal Perl structure of the event.

NOTE

These variables cannot be customized unless the method of event reception is changed.

Single variables

The INPUT_VARIABLES section begins with a line that contains only the keyword INPUT_VARIABLES and ends with a line that contains only the keyword **END**. Within the section each variable is declared in a single line. The name of a variable must begin with the **$** character.

Variables contain scalar values such as $avariable. These values are scalar in the context of the Perl language. They can represent a string or a number, depending on the variable definition. Therefore, these variables must be assigned only to relevant BMC Impact Manager slots.

The variables also can be references in Perl context to any type of object. Such reference variables cannot be used for slot assignment, except as references to a list as discussed below, but can be used as arguments of customized functions. If the variable is a reference to a Perl list, it can be an assignment value for a slot which is of the LIST_OF type.

Series of variables

The MAP file supports declaration of series of variables. The syntax is the same as for scalar variables except that the variable name *must* be followed by an interval of values. Three forms of acceptable declarations of sets are:

```
$setA 0–43
$setB 43
$setC 0–i
```

The first line declares 44 scalar variables, `$setA1`, `$setA2`, . . . , `$setA43`. The second line is equivalent to the first declaration. The third line declares a set of undefined size. In this case, variables `$setC0`, `$setC1`, `$setC2`, ..., `$setC99` can be used in the MAP file. If the set has a size larger than 100, its size must be specified in the declaration. There is no size limit to sets other than your computer's memory resources. In other words,

```
$set N
```

is equivalent to

```
$set0
$set1
...
$setN
```

Order

Except in the variables declaration, the input variables section can contain comments. This section must be the first, as illustrated in the example of the input variables section of the SNMP Trap Adapter in Figure 3.

Figure 3 Comments code example (part 1 of 2)

```
INPUT_VARIABLES
    $Version
    $Community
    $Enterprise
    $TrapType
    $Specific
    $TimeTicks
    $SourceAddr
    $RequestId
    $ErrorStatus
    $ErrorIndex
    $EnterpriseName
    $TrapTypeStr
# Special INPUT_VARIABLES (references on list)
    $var
    $oid
```

Figure 3 Comments code example (part 2 of 2)

```
    $oidt
# Variable for indexes. $idx is a reference on a list # of lists.
# It cannot be used for direct slot assignments
    $idx
END
```

The MAP file is case sensitive. `$Source` and `$SOURCE` are two different variables.

For backward compatibility, the `$LOGFILE INPUT_VARIABLES` contains the value of the `LogLOGFILE` parameter in the configuration file or "`logfile`" as default. A new `INPUT_VARIABLE`, `$logname`, is available and contains the name of the log file. In addition, the syntax of the parameters is changed to be consistent with all Adapters. Specific parameters to the LogFile Adapter begin with `Log` in which the **L** is upper case.

Slot assignments

Most MAP file entries are either a slot assignment or a condition. A slot assignment is used to enter an input value into a BMC Impact Manager slot.

Unlike variables, the BMC Impact Manager slots are not declared in the MAP file. They are syntactically reproduced when they are sent to the cell. Figure 4 illustrates acceptable slot assignment formats:

Figure 4 Slot assignment formats

```
slot_name = 1234
slot_name = "a string"
slot_name = #mydefine
slot_name = $variable
slot_name = $variable[3]
slot_name = buildlist (<args>, ...)
slot_name = gethostbyaddr (<arg>)
slot_name = gethostbyname (<arg>)
slot_name = lower (<arg>)
slot_name = printf (<format>, <args>, ...)
slot_name = substr (<arg>, <start_pos>, <length>)
slot_name = upper (<arg>)
slot_name = <perl_func> (<args>, ...)
```

The first five lines are simple assignments either from constants (lines 1-3) or from a variable (lines 4 and 5). When `slot_name` is assigned to a constant, this constant is recognized from the first non-space character after the equal (=) sign, to the last non-space character on the same line.

Strings must be delimited by double quotes (“”). Because of the construction of a MAP file, a string cannot contain new lines. In order to include new lines in a string, a user-defined Perl function must be defined. For more detail, see the last assignment for `slot_name` containing `<perl_func>`. Strings can contain double quotes since the last double quote in the line is considered to be the end of the string.

Line 5 involves a substring of a variable, `$variable`. This corresponds to the matching substring of `$variable`, evaluated in a condition statement involving a regular expression matching `$variable`.

Slots can also be assigned to the return value of one of the seven functions as listed in Table 11. Functions are used to insert in a slot a value held by an entity in the network, such as the IP address of a computer.

Table 11 Slot Functions

Function	Description
`buildlist (<args>, ...)`	`<args>` is a list of arguments represented by constants or variable names. `buildlist` returns a list of values that can be assigned to a `LIST_OF` slot. See the *BMC Impact Manager Knowledge Base Reference Guide* for details about `LIST_OF` slots. If the left side of the `buildlist` assignment is not a `LIST_OF` slot, the cell will reject the event.
`gethostbyaddr (<arg>)`	`<arg>` is either a string or a variable name containing the internet address of a computer, such as `123.456.789.012`. `gethostbyaddr` returns the full name of the computer, or zero (`0`) if not found.
`gethostbyname(<arg>)`	`<arg>` is either a string or a variable name containing the name of a computer. `gethostbyname` returns the internet address of the computer in the form a string, such as `123.456.789.012`, or zero (`0`) if not found.
`lower (<arg>)`	`<arg>` is either a string or a variable name containing a string. The function returns `<arg>` converted to lowercase.
`substr (<arg>, <start_pos>, <length>)`	`<arg>` is either a string or a variable name containing a string. `<start_pos>` and `<length>` are either integers or variables containing integers. `substr` acts as the Perl `substr` function. See Perl documentation for details.
`printf (<format>, <args>, ...).`	`<format>` is either a string or a variable name containing a string. `<args>` is a list of arguments, possibly empty, represented by constants or variables, similar to other assignments, whose values must correspond to the `<format>` argument. `printf` behaves like the `printf` Perl function and therefore like the C `printf` function. See C and Perl documentation about `printf` for details.
`upper (<arg>)`	`<arg>` is either a string or a variable name containing a string. The function returns `<arg>` converted to uppercase.

The most general method of assignment is represented by the following assignment form: slot_name can be assigned to the return value of built-in Perl functions, or any user defined Perl function, that is defined in an external file and loaded with the REQUIRE directive.

```
slot_name = <perl_func>( <args>, ...)
```

<perl_func> can be either a built-in Perl function, such as length or time, or a function defined in files loaded with the REQUIRE directive. The arguments <args> of <perl_func> can be any of the following:

- constants such as "a string", 1234
- input variables such as $avariable, $anothervar[1]
- variables known in the global context such as @INC, $ENV{'PATH'}
- simple expressions involving function calls such as length($avariable)

Constants and input variables are used as in other assignments. In addition, <perl_func> can accept other arguments that are valid Perl expressions in the global context. Local variables of the MAP module MA::Map and other modules are not known, except for variables with a global scope not declared with a "my' statement, when their name is preceded by the module name and ::). These expressions cannot contain comma characters (,), except in strings that have single or double quotes. For example, an argument such as badarg($farg1, $farg2) is not valid, while goodarg("string, string") is a valid argument.

Conditions

A condition is a Boolean expression associated with a CLASS statement that must be satisfied so the current event matches the class. A condition has the format:

```
<var> <operator> <comp_value>
```

<var> is either an input variable or a substring of an input variable set in a previous condition, such as:

```
$avariable
$anothervar[1]
```

<operator> is one of the operators listed in Table 12.

Table 12 Operators

Operator	Description
equals	When <var> and <comp_value> contain numbers, the condition is satisfied if <var> and <comp_value> are equal. When <var> and <comp_value> contain strings, the condition is satisfied if <var> and <comp_value> contain the same string. Remember that these are case sensitive. When <var> contain a string and <comp_value> is a Perl regular expression, the condition is satisfied when <var> matches with the Perl regular expression.
not_equals	not_equals can be used in the same conditions as equals and returns the negation of the expression where not_equals is replaced by equals.
smaller	When <var> and <comp_value> are numbers, the condition is satisfied if <var> is strictly smaller than <comp_value> (<var> < <comp_value>).
greater	When <var> and <comp_value> are numbers, the condition is satisfied if <var> is strictly larger than <comp_value> (<var> > <comp_value>).
smaller_or_equals	When <var> and <comp_value> are numbers, the condition is satisfied if <var> is smaller or equal to <comp_value> (<var> _ <comp_value>).
greater_or_equals	When <var> and <comp_value> are numbers, the condition is satisfied if <var> is larger or equal to <comp_value> (<var> _ <comp_value>).

When the operator is smaller, greater, smaller_or_equals or greater_or_equals, <var> must contain a number.

<comp_value> represents:

- number, such as 12, -2, 0xFF, .314E+2.
- string, such as "A string" (<operator> must be equals or not_equals).
- regular expression, such as /.*matchthis.*/ (<operator> must be equals or not_equals).
- DEFINE representing one of the above three values, such as #mydefine.

Pattern matching

<comp_value> regular expressions are evaluated with the Perl rules for pattern matching. See your Perl documentation about pattern matching for details. The regular expression can contain parentheses that match substrings of <var>. These substrings can be accessed by using the [] operator on <var>. Technically, if <var> represents $var, the Perl variables $1, $2, ..., $9 are stored in $var[1], $var[2], . . ., $var[9]. For example:

```
$match equals /\s*(\w+)\s+(\w+)\s+.*/
$match[1] equals "first"
slot = $match[2]
```

Suppose that $match contains "first second third etc...." The first condition is satisfied since $match contains at least two words. After the first condition, $match[1] contains "first" and $match[2] contains "second". The second condition is satisfied and slot is set to "second".

A more complex example is:

```
$match equals /\s*(\w+)\s+(\w+)\s+.*/
$match[1] equals /f(..)(.*)/
slot = $match[2]
```

The second condition also involves pattern matching. The second condition is satisfied; $match[1] contains "ir" and $match[2] contains "st". slot is set to "st". Such substrings can be used in nested CLASS structures. The second example shows how a substring, itself, can be used as an expression for pattern matching. In this case, the new partial strings extracted from the subexpression are accessed by using the operator [] from the base string. To clarify, examine the condition: $match[1] equals /f(..)(.*)/. After a successful matching, the expressions corresponding to the two expressions (..) and (.*) will be stored in $match[1] and $match[2], which overrides the previous values of $match[1] and $match[2]. Expressions such as $match[1][1] and $match[1][2] are *not* allowed. This may seem counterintuitive, but it is very much in accordance with the Perl approach to pattern matching, where special variables $1, $2, . . ., take new values at each match.

Regular expressions

- cannot contain references to the variables declared in the INPUT_VARIABLES section
- support the following modifiers: g i m o s x

Consult the Perl documentation for details about such modifiers. For example.

```
$match equals /AnY_cAsE/i
```

matches the string "any_case" in lower and upper case.

INIT section

The INIT section contains assignments of BMC Impact Manager slots that are common to all events. Figure 5 on page 53 illustrates the structure of the INIT section.

Figure 5 INIT structure

```
INIT
    <assignment>
    <assignment>
    ...
END
```

<assignment> is an assignment. When an external event is mapped into a BMC Impact Manager event, the assignments of the INIT section are evaluated for each event. A CLASS or DEFAULT assignment overrides a slot value previously set in the INIT section.

DEFAULT section

Figure 5 illustrates the structure of the DEFAULT section.

Figure 6 DEFAULT structure

```
DEFAULT <class_name>
    <assignment>
    <assignment>
    ...
END
```

<class_name> is the name of the default class, such as EVENT, DEF_ADAPTER_EV. Be certain that the class name is recognized by the cell to which it is sent. If an event does not match any CLASS requirements, an event of class <class_name> can be generated. The slots of the event are set by evaluation of the assignments of the INIT and DEFAULT sections. DEFAULT assignments override INIT assignments of the same slot. The DEFAULT section can contain no assignments. The DEFAULT section is optional. At most, one DEFAULT section is authorized.

By default, the DEFAULT mechanism is disabled. That is, if an event does not match any class, it is dropped before being sent to the cell. To enable the DEFAULT class mechanism, either:

- Use the **-e** command-line option without argument, or
- Use the MapUseDefaultClass (MapUseDefaultClass=1) parameter in the configuration file (**etc/*.conf**).

CLASS section

The CLASS structures contain the main information about how to map Adapter events into BMC Impact Manager events. CLASS structures can be nested. Figure 7 on page 54illustrates the CLASS leaf section.

Figure 7 CLASS leaf structure

```
CLASS <class_name>
<condition>
<condition>
...
<assignment>
<assignment>
...
END
```

General structure

Figure 8 illustrates the general structure for nested class mapping.

Figure 8 Nested class mapping Structure

```
CLASS <class_name>
   <condition>
   <condition>
   ...
   <assignment>
   <assignment>
   ...
   CLASS <sub_class_name>
      <condition>
      ...
      <assignment>
      ...
      CLASS <subsub_class_name>
      ...
      END
   END
   CLASS <sub_class_name>
   ...
   END
END
```

<class_name>, <sub_class_name>, and <subsub_class_name> are names of the BAROC classes. An event matches a CLASS <class_name> when all conditions (<condition>) associated with that class and parent classes are satisfied. If all conditions (<condition>) are satisfied, assignments (<assignment>) associated with that CLASS and possible parent classes (CLASS) are evaluated in order to build a BAROC event. This BAROC event is then sent to a cell. Classes are evaluated sequentially. As soon as an incoming event satisfies all conditions of a CLASS and does not match any subclass of that CLASS, an event of the corresponding BAROC class is created, then sent.

Even if it is often the case, the CLASS structure of the MAP file is not required to reflect the BAROC classes as defined in a **.baroc** file. The nature of the hierarchical classes in the MAP file is purely syntactic. The BAROC class corresponding to a subclass need not be a subclass, in the BAROC sense, of the BAROC class corresponding to the parent CLASS in the MAP file sense. In addition, a MAP file can contain several classes having the same <class_name>.

Class name

Moreover, a MAP file can contain CLASS <class_name> statements of nonexistent corresponding BAROC class names (<class_name>). In this case, either the MAP writer has verified that no event will match that CLASS without matching any subclass or events of CLASS <class_name> will be dropped by the cell, as in Figure 9 on page 55.

Figure 9 Class name example

```
CLASS A
   $var1 equals /string3/
   slot1 = “abc”
   CLASS B
      $var2 equals /string2/
      slot2 = “def”
   END
END
CLASS B
   $var2 equals /string3/
   slot1 = “ghi”
END
CLASS dummy
   $var1 equals /string3/
   CLASS A
      $var1 equals /string4/
      slot1 = “jkl”
   CLASS B
      $var2 equals /.*/
      slot1 = “mno”
   END
END
```

The above CLASS structure defines several possible matches for CLASS A or CLASS B. Even if CLASS B is nested in CLASS A in the first CLASS A structure, CLASS B is not necessarily a BAROC subclass of CLASS A. Note that the BAROC CLASS B must have a defined slot, slot1. CLASS dummy need not be a defined BAROC class since the condition under the nested CLASS B, $var2 equals /.*/, is always satisfied, which means that it is impossible to generate a dummy event from that MAP file sample.

Directives

A MAP file can also contain a DEFINE or a REQUIRE directive.

DEFINE directive

A DEFINE directive is a line beginning with the DEFINE keyword, as the following illustrates.

```
DEFINE the_define “define value” <new line>
```

When such a statement is made, each subsequent statement in which a value is represented by #the_define (the first ’#’ character is required), is replaced by “define value,” excluding the space characters before the end of line.

A comment line is a line in which the first non-space character is a **#**, a **!**, or a **%** character. Comment lines and empty lines are ignored.

REQUIRE directive

Instead of using the MapFunctionsName parameter, user-defined Perl files can be included in the **.map** file by using the REQUIRE directive, as follows:

```
  REQUIRE <perl_module>
```

In general, <perl_module> is a Perl module or script containing functions accessible from within the **.map** file. The location must be either the Perl standard library directory or in the ***MCELL_HOME*\lib\perl** directory. Subdirectories are allowed with the standard Perl “: :“ notation. For example, Figure 10 is an extract from the **mcsyslogd.map** file.

Figure 10 mcsyslogd.map file excerpt

```
REQUIRE MA::MapUtil
# ... lines are missing
CLASS SYSLOG_BASE $complete equals
   logfile = $LOGFILE
   source = "Syslog"
   mc_incident_time = DateSyslog2Epoch ($complete[1])
```

DateSyslog2Epoch is defined in *MCELL_HOME***lib\perl\MA\MapUtil.pm.**

Multiple REQUIRE statements are allowed within the same **.map** file.

Following the object-oriented Perl methods, the first argument of each user-defined call is a reference to the `MA::Map` object. The `INPUT_VARIABLES` of the MAP file can be found in the `{specific_event}` key of the `MA::Map` object, as Figure 11 illustrates.

Figure 11 Map call

```
sub DateSyslog2Epoch
   {
      my MA::Map $map = shift;
      my $str = shift;
# See MapUtil.pm for the rest of source.
#
# INPUT_VARIABLES could accessed as (for instance):
# my $ivars = $map->{specific_event};
# foreach my $iv (keys %$ivars)
# {
# # Do something with $iv and $ivars > {$iv}
```

In Figure 11, above, `DateSyslog2Epoch` has been called with the single argument `$complete[1]`. The Adapter automatically adds the `MA::Map` object to the beginning of function arguments, so that the first shift inside the implementation of `DateSyslog2Epoch` returns the MAP object, while the second shift returns the string corresponding to `$complete[1]`. See the ***MCELL_HOME*\lib\perl\MA\Map.pm** file for more details.

NOTE

The function that `$complete` performs in the LogFile, Perl EventLog for Windows, and SNMP Adapter mappings is performed by `$data` for the IP Adapters (see references to the `$data` variable in "IP Adapter parsers" on page 101). `$data` is provided automatically. However, you cannot rename it.

Chapter 4

Using the BMC Impact Event Adapters

This chapter discusses the following topics:

Enabling and disabling BMC Impact Event Adapters 59
Starting and stopping the BMC Impact Event Adapters 61
- Starting the engine manager process from the command line 61
- Starting the Adapter processes as services 63
- Restriction for BMC Impact Event Adapters under MS Windows 63
- Stopping the BMC Impact Event Adapters 63
- Using instance control to start and stop Adapters 64
- Restarting an Adapter after modifying the MAP file. 67

Enabling tracing 67
- Trace files 67
- Trace levels 68
- Errors and Adapter events 68

Adapter status events 69

Enabling and disabling BMC Impact Event Adapters

Before you can start the various BMC Impact Event Adapters (either by starting them as services or starting the engine manager process), you must define and enable the Adapter that you want to run in the **mcxa.conf** file.

TIP

If you want enable or disable an Adapter instance during run-time, see "Using instance control to start and stop Adapters" on page 64.

To enable an Adapter instance

1 In a text editor, open the **mcxa.conf** file.

This file is located in one of the following directories:

- On Windows: ***MCELL_HOME*\etc**
- On UNIX: ***MCELL_HOME*/etc/**

2 In the **mcxa.conf** file, navigate to the definition of the Adapter instance that you want to enable.

3 Remove or comment out the word **DISABLE** from the Adapter definition.

4 Repeat step 3 for each Adapter instance that you want to enable.

5 Save and close the file.

Within a minute or two, a status message indicating the current state of the newly enabled Adapter is sent to the cell server and is displayed in the BMC Impact Explorer Console.

To disable an Adapter instance

1 In a text editor, open the **mcxa.conf** file.

This file is located in one of the following directories:

- On Windows: ***MCELL_HOME*\etc**
- On UNIX: ***MCELL_HOME*/etc/**

2 In the **mcxa.conf** file, navigate to the definition of the Adapter instance that you want to disable.

3 Add the word **DISABLE** to the Adapter definition or if **DISABLE** was commented out, remove the comment character (**#**).

4 Repeat step 3 for each Adapter instance that must be disabled.

5 Save and close the file.

Within a minute or two, a status message indicating the current state of the newly disabled Adapter is sent to the cell server and is displayed in the BMC Impact Explorer Console.

Starting and stopping the BMC Impact Event Adapters

You can use the following methods to start the various BMC Impact Event Adapters:

- The Adapters that use the engine manager (LogFile Adapter, SNMP Adapter, and Perl Eventlog Adapter for Windows) can be started as follows:

  — On Microsoft Windows computers, you start the engine manager (mxca process) from the command line or from the Services window.

  — On UNIX computers, you start the engine manager (mxca) process from the command line.

  For instructions, see "Starting the engine manager process from the command line" and "Starting the Adapter processes as services" on page 63.

- The IP Adapters (on Windows and UNIX) can be started and stopped individually after the primary engine manager process is running. For more information, see "Using instance control to start and stop Adapters" on page 64. When running on Microsoft Windows computers, all Adapters that use the engine manager can be started from the command line or from the Services window.

Starting the engine manager process from the command line

At a command prompt, run the **mcxa.cmd** (Windows) or **mcxa.sh** (UNIX) executable to start the engine manager (mcxa) process.

On Windows, the **mcxa.cmd** command starts the appropriate services. The services were created during product installation.

Table 13 lists the command options.

Table 13 mcxa command options (part 1 of 2)

Option	Description
-c	alternate configuration file Default: *MCELL_HOME***etc****mcxa.conf**
-d	Debug (prevents daemonization), available only on UNIX platforms
-h	help

Table 13 mcxa command options (part 2 of 2)

Option	Description
-n	Send all events to a specific cell Use this option to specify the cell to which you want to connect. Use one of the following option formats: ■ Designate a cell name by specifying **–n *cellname***. This format option maps the specified cellname to the host, port, and encryption key by looking the values up in the **mcell.dir** file. ■ Designate a host, port, and encryption key by using **–n *@host:port#key*.** The variable *host* represents either a host name or an IP address value; *port* represents the port number value, and *key* represents the encryption key value. You can specify the designation to accept **–n *@host:port*** or **-*@host*** and accept the default values for key(**0**) and port (**1818**). This format uses the specified host, port, and encryption key to connect to the cell without looking up information in the **mcell.dir.**

-t	Specifies trace file and / or level, such as 1–6
	
	Use `-T` for long headers.
	
	You can make minor changes to the command syntax to modify how debug output is displayed or stored.
	
	■ Use a single colon (:) in the command to send output to the default trace file, *MCELL_HOME***tmp****Adapters****mcxa.trace**.
	
	Example:
	
	`mcxa.cmd -t:6`
	
	■ Use a double colon (::) in the command to display output on-screen (stdout).
	■ Use a single colon (:) in the command to send output to the specified trace file.
	
	Example:
	
	`mcxa.cmd -t \tmp\mytracefile.txt:6`
	
	starts the engine manager at trace level 6 and use **tmp****mytracefile.txt** trace file.

-z	displays the Adapter version

NOTE

Command line interface options take priority over options in the **mcxa.conf** file.

Starting the Adapter processes as services

1 Choose **Start => Settings => Control Panel => Administrative Tools => Services.**

2 From the list of services, select **BMC Impact Event Adapters**.

3 From the menu bar, choose **Action => Start**.

The Adapters are running when the Status column of the Services window displays **Started** for the Adapters.

Restriction for BMC Impact Event Adapters under MS Windows

Under MS Windows, you can start the Logfile Adapter as a service only if all the enabled Logfile Adapters are pointing to files on local drives. If any LogFile Adapter is pointing to a file on a remote system or on a mapped drive, then you must run the mxca.cmd from the command line to start the mcxa process. This restriction also applies to uniform naming convention (UNC) paths such as \\acomputer101\fileshare\file because you cannot provide login credentials to access the path from the service.

Stopping the BMC Impact Event Adapters

You can stop the Adapters by using one of the following methods:

- On Windows, stop the **BMC Impact Event Adapters** service from the Services window.

  This method sends an **MC_ADAPTER_STOP** event before the Adapters stop.

- On UNIX, stop the Adapters by using either the **kill** command or a shell script, such as the **mcxa** script located in **etc/init.d**.

  This method sends an **MC_ADAPTER_STOP** event before the Adapters stop.

NOTE

Do not use the **kill -9** command to stop the Adapters unless they are in an infinite loop. Use the regular **kill** command, instead.

- On UNIX or Windows, create a file called **mcxa.stop** and add it to the MCELL_HOME**/etc/** directory**.**

  When this file is added to the MCELL_HOME**/etc/** directory, the Adapters stop.

  The contents of the **mcxa.stop** file are not important. When the Adapter detects the presence of the file, normally within five seconds, it deletes the file and then stops. This method sends an MC_ADAPTER_STOP event before the Adapters stop.

  If you used the -c option with the **mcxa.sh** command or **mcxa.cmd** command to specify a configuration file other than **mcxa.conf**, the stop file must have the same primary name as that configuration file, using the **.stop** extension. For example, if your Adapters configuration file is **adap.conf**, name the stop file **adap.stop**.

NOTE

If you stop and start the BMC Impact Event Adapters service in quick succession, or use the restart option in the service manager, you might see the following error messages in the BMC Impact Explorer console:

```
Couldn't be an UDP server on port 16

MA: EngineMgr: Couldn't be an UDP server on port 162
```

If you see these messages, wait a short time until the expected stop messages appear before restarting the Adapters. For example, with an SNMP Adapter enabled, wait until the messages `BMC Impact Event Adapter stopped` and `Adapter Snmp (Engine: 'MA:ESnmpTrap') stopped by 'mcxa'` appear before restarting the BMC Impact Event Adapters service.

Using instance control to start and stop Adapters

The **mcxactrl.pl** instance control command enables you to perform various operations on Adapter instances, including starting and stopping specific Adapter instances. The **mcxactrl.pl** command communicates with the engine manager using a control port opened by the engine manager. The port accepts only local host connections.

WARNING

Use the CtrlPort Adapter with caution. It is safe to use only on computers where log-ins are restricted to personnel who are authorized to control mcxa.

CtrlPort refuses connections from any address other than 127.0.0.1 (localhost), but this is not adequate security for computers that allow people who are not authorized to control the mcxa process to log on.

Figure 12 shows the syntax of the **mcxactrl.pl** command.

Figure 12 mcxactrl.pl command syntax

```
mcxactrl.pl [ --debug ] [ --mgr managername ] --port number [--help | -h | --list
| --start name | --stop name | --stopmgr ]
```

Table 14 on page 65 lists the options that you can with the **mcxactrl.pl** command. You can specify only one option each time the command is run.

NOTE

Double hyphens must precede the options of the **mcxactrl.pl**command.

Table 14 mcxactrl.pl command options

Parameter	Description
`--debug`	activates debugging of the procedure that is specified (starting, stopping, listing, and so on). Optional. For more information, contact BMC Software support.
`--mgr`	engine manager affected by the command. Optional. Default: **mcxa**, the default specified in the **mcxa.conf** file
`--port`	control port on which the engine manager listens. Required - if the port number has been changed from 1998 (in **mcxa.conf**).
`--help`	prints usage information and exits. Optional.
`--list`	lists all Adapters and their current status. Optional.
`--start` *instancename*	starts the specified Adapter instance. For more information, see “Starting and stopping an Adapter without stopping the engine manager” on page 66. Optional.
`--stop` *instancename*	stops the specified Adapter instance. For more information, see “Starting and stopping an Adapter without stopping the engine manager” on page 66. Optional.
`--stopmgr`	stops the engine manager specified with `--mgr`

Starting and stopping an Adapter without stopping the engine manager

You can use the **mcxactrl.pl**command to start or stop an individual enabled Adapter without stopping the engine manager on which it is running or other Adapters running on that engine manager. To stop an Adapter, the Adapter must be enabled, and the engine manager must already be running.

The **mcxactrl.pl** command can only be performed locally.

To start or stop an Adapter by using the mcxactrl.pl command

1 Using a text editor, open the **mcxa.conf** file.

2 Navigate to the CtrlPort Adapter definition and complete the following steps:

A *(optional)* If you intend to use a non-default port value in the **mcxactrl.pl** command, replace the default LocalPort value with the required port number.

When executing the **mcxactrl.pl** command in step 4, the port number you specify must match the LocalPort value recorded in the CtrlPort Adapter definition in the **mcxa.conf** file. The default port number recorded in the CtrlPort Adapter definition in the **mcxa.conf** file is 1998.

B Enable the CtrlPort Adapter by removing or commenting out the word **DISABLE** from its definition.

NOTE

After the CtrlPort Adapter is enabled, you can run the **mcxactrl.pl** command without re-enabling the CtrlPort Adapter.

3 Save and close the **mcxa.conf** file.

4 On the command line, enter the following **mcxactrl.pl**instance control command:

```
mcxactrl.pl --mgr managername --port number { --start | --stop }
AdapterName
```

NOTE

Double hyphens, not single hyphens, must precede the arguments in the **mcxactrl.pl**start and stop commands.

For information about additional command line options, see Table 14 on page 65.

Restarting an Adapter after modifying the MAP file

If you modify the contents of a MAP file, you must stop and restart any Adapters that are using that MAP file to apply the changes.

Enabling tracing

You can enable tracing in one of the following ways:

- by defining the `TraceFile`, `TraceLevel`, and `TraceFileCount` global parameters in the **mcxa.conf** file

  Additional global parameters also affect tracing. For more information, see "Global parameters" on page 30.

- by using the **mcxa.cmd** or **mcxa.sh** commands.

  Table 13 on page 61 lists the command line options that you can use to enable tracing with the **mcxa.cmd** or **mcxa.sh** commands.

Trace files

By default, the mcxa process trace file is ***MCELL_HOME*\tmp\Adapters\mcxa.trace**. Each time the Adapter is started, a new trace file is created and old logs are rotated.

EXAMPLE

mcxa.trace is rotated to **mcxa0.trace**
mcxa<n>.trace is rotated to **mcxa<n+1>.trace**

You control the number of old log files through the `TraceFileCount` parameter setting in the **mcxa.conf** file. A value of **0** (zero) removes all old trace files. For more information and default values, see the header of the **mcxa.conf** file.

You can specify a different trace file in a different location by using the **mcxa.cmd** or **mcxa.sh** commands. For more information about command line options, see Table 13 on page 61.

Trace levels

Use the `TraceLevel` global parameter in the **mcxa.conf** file to specify the trace level that you want to use. This setting will apply to all Adapters running in the Adapter framework. You can specify the following trace levels:

- **0** – disables all traces
- **1** – header messages
- **2** – fatal messages
- **3** – major errors
- **4** – minor errors
- **5** – normal output
- **6** – verbose output

NOTE

`TraceLevel=6` produces very large trace files and is recommended only for debugging.

Errors and Adapter events

In addition to logging errors to a log file, Adapters send the cell error messages and notifications in the form of events in the following situations:

- When an Adapter starts, it sends an **MC_ADAPTER_START** event.
- When an Adapter stops, it sends an **MC_ADAPTER_STOP** event.
- When the Adapter encounters a major or fatal error in the configuration file, the Adapter reports the error in the trace file according to the trace level. When the error does not come from the configuration file, the Adapter reports the error to the cell whose name corresponds to the `ServerName` parameter of the `[default]` (unnamed) section of the configuration file. The event class is **MC_ADAPTER_ERROR**.

Error and message events sent by the IP Adapters are described in "Adapter status events" on page 69.

Adapter status events

When an Adapter instance experiences a change in status, the instance sends a status report to the cell, formatted as an event. You can view these events in the Adapters collector in the BMC Impact Explorer events view. Table 15 describes the status events.

Table 15 IP Adapter status events (part 1 of 2)

Status event	Slots	Description
MC_ADAPTER_START	■ msg = "Incoming connection from *ipaddress*:*port* (*host*)" ■ mc_object = "*ipaddress*:*port*" ■ mc_object_class = "CLIENT"	sent when a command from **mcxactrl.pl** starts an Adapter instance
MC_ADAPTER_STOP	■ msg = "Removing connection from *ipaddress*:*port* (*host*)" ■ mc_object = "*ipaddress*:*port*" ■ mc_object_class = "CLIENT"	sent when a command from **mcxactrl.pl** stops an Adapter instance
MC_ADAPTER_STOP	■ msg = "Client hungup: destroying connection from *ipaddress*:*port* (*host*)" ■ mc_object = "*ipaddress*:*port*" ■ mc_object_class = "CLIENT"	sent when **mcxactrl.pl** terminates a control connection
MC_ADAPTER_START	■ msg = "Adapter '*Adapter*' (Engine: '*engine*') started by '*module*'" ■ mc_origin = "*Adapter*" ■ mc_origin_class = "*engine*" ■ mc_object = "*Adapter*" ■ mc_object_class = "*engine*" ■ mc_parameter = "STATUS" ■ mc_parameter_value = "STARTED"	sent when an Adapter instance starts
MC_ADAPTER_STOP	■ msg = "Adapter '*Adapter*' (Engine: '*engine*') stopped by '*module*'" ■ mc_origin = "*Adapter*" ■ mc_origin_class = "*engine*" ■ mc_object = "*Adapter*" ■ mc_object_class = "*engine*" ■ mc_parameter = "STATUS" ■ mc_parameter_value = "STOPPED"	sent when an Adapter instance stops
MC_ADAPTER_ERROR	■ msg = "*last_error_discovered*" ■ severity = "CRITICAL" ■ mc_origin = "*origin*" ■ mc_origin_class = "*origin_class*" ■ mc_object = "*object*" ■ mc_object_class = "*object_class*" ■ mc_parameter = "STATUS" ■ mc_parameter_value = "ERROR"	sent when an IP Adapter experiences a critical error

Table 15 IP Adapter status events (part 2 of 2)

Status event	Slots	Description
MC_ADAPTER_START	msg = "BMC Impact Event Adapter started"	sent when an engine manager initializes
MC_ADAPTER_STOP	msg = "BMC Impact Event Adapter stopped"	sent when an engine manager exits

Chapter 5

Using the SNMP Adapter Configuration Manager

This chapter describes the procedures related to using the SNMP Configuration Manager product and installing the BAROC files in the Knowledge Base of the cell.

Accessing the Web-based interface 71
Publishing MIB files 72
Viewing or editing the MAP file 74
Unpublishing MIB files. 76
Installing the BAROC files in the Knowledge Base of a cell 77

Accessing the Web-based interface

To access the Web-based interface of the SNMP Configuration Manager, go to **http://*hostname*:*port*/snmpadapter** or **http://*ipaddress*:*port*/snmpadapter**.

- *hostname* is the computer where you have installed the BMC Impact Event Adapters and the SNMP Configuration Manager.
- *ipaddress* is the IP address of the computer where you have installed the BMC Impact Event Adapters and the SNMP Configuration Manager.
- *port* is the port you have configured for the SNMP Configuration Manager.

EXAMPLE

For example, if an Apache Tomcat server is installed at a listening port as a part of the SNMP Configuration Manager installation, you must first launch the Tomcat server using the following command:

```
%MCELL_HOME%SNMPAdapter\run.bat
```

After running this command, launch the SNMP Configuration Manager using the following URL:

http://tomcathost:8080/snmpadapter

where **tomcathost** is the name of the host and the default listening port is **8080**.

NOTE

The first time that you access the Web-based interface of the SNMP Configuration Manager on a Windows 2003 computer, you could get a security dialog box indicating that the content from **http://*hostname*** has been blocked. To access the web-based interface of the SNMP Configuration Manager, you have to first add **http://*hostname*** to the list of trusted sites.

The interface displays the following links in the navigation bar:

- **Publish MIBs**
- **View / Edit MAP**
- **Unpublish MIBs**

Publishing MIB files

Publishing MIB files is the process of converting information from the MIB files into cell classes.

To publish MIB files

1. Click **Publish MIBs** in the navigation bar.
2. Click **Browse** to select a MIB file that is accessible from your local computer. You must specify at least one MIB file.

You can use the **Add** and **Remove** buttons to create a list of MIB files to be sent for publishing. The add and remove actions do not affect the files saved in the **net-snmp** directory on the BMC Impact Event Adapters.

- To publish more than one MIB file at a time, click **Add**.

  A new row is added below the existing row.

  You can simultaneously publish a maximum of 10 MIB files.

- To remove a MIB file from the list, select the check box next to that file, and then click **Remove**.

3 To publish the MIB files, click **Publish MIBs**.

You are prompted to restart the BMC Impact Event Adapters.

4 Click **Yes** to restart the BMC Impact Event Adapters.

The BMC Impact Event Adapters will start receiving events from the devices whose MIB files you have published only after you restart the BMC Impact Event Adapters. Click **No** if you do not want to restart the BMC Impact Event Adapters at this point.

You can view the status of the publish MIBs process in the Publishing Messages area. The message `MIB File: mibFileName is not proper` might be displayed in the Publishing Messages area. *mibFileName* is the name of the MIB file that is not proper. To identify the exact reason for this message, you must check the output of the **mib2map.pl** utility, which is also displayed in the Publishing Messages area.

The message `MIB File: mibFileName is not proper` is displayed due to one of the following reasons:

- The MIB file is not valid.
- The MIB file is dependent on another MIB file that does not exist in the **Net-SNMP** directory.
- The MIB file contains dependencies that do not correspond to the information that is available in the parent MIB file.

NOTE

If one or more of the selected MIB files already exist in the **Net-SNMP** directory, you are prompted to overwrite the MIB files.

- If you choose to overwrite the MIB files, the existing MIB files in the **Net-SNMP** directory are replaced with the new MIB files.
- If you choose not to overwrite the MIB files, only the MIB files that do not already exist in the **Net-SNMP** directory are published.

For more information about the steps involved in the publish MIBs process, see Appendix A, "Backend processes for publishing and unpublishing MIB files."

Viewing or editing the MAP file

You can use the **View / Edit MAP** link in the navigation bar to browse individual event classes in the **mcsnmptrapd.map** file. The event classes are displayed in a tree structure. You can select an event class, and then add new variables, or edit and remove existing variables from that event class. The modifications that you make in the **mcsnmptrapd.map** file are not reflected in the corresponding MIB file.

NOTE

Each time you edit the MAP file, a backup of the previous version is saved in the ***MCELL_HOME*\tmp\Adapters\snmpAdapter\map** directory. The naming convention for the backup files is **mcsnmptrapd_*mm*Month_*dd*Day_*yyyy*Year_*hh*Hrs_*mm*Mins_*ss*Secs.map**.

You can add, edit, or remove the following variables from the **mcsnmptrapd.map** file:

- `msg`
- `mc_tool_class`
- `mc_tool`
- `mc_host_address`
- `mc_location`
- `severity`

  The values for this variable are UNKNOWN, OK, INFO, WARNING, MINOR, MAJOR, CRITICAL, and DOWN.
- `mc_priority`

  The values for this variable are PRIORITY_5, PRIORITY_4, PRIORITY_3, PRIORITY_2, and PRIORITY_1.
- `mc_notes`

WARNING

When you click **View / Edit MAP** in the navigation bar, the data for the event classes is copied from the server to your local computer. You can then modify the data for the event classes on your local computer. However, the changes are saved on the server only when you click **Update MAP File**.

When you retrieve the event classes data from the server, the **mcsnmptrapd.map** file is not locked. While you are modifying the **mcsnmptrapd.map** file, if the file is modified in another instance and the changes are saved on the server, you will not be able to save the changes you have made to the **mcsnmptrapd.map** file.

To edit event classes

1 Click **View / Edit MAP** in the navigation bar.

2 Select the event class you want to edit.

3 Perform the following actions.

- To add a variable, click **Add New**. In the Add New Variable dialog box, select the variable, and enter or select the value for that variable.
- To edit a variable, click the value of that variable. In the Update Value dialog box, modify the value.
- To remove a variable, select the check box next to that variable, and click **Remove**.

4 To save the changes, click **Save**.

The changes made to the event class are saved on your local computer. You can edit more than one event class before you save the changes on the server.

5 To save your changes on the server, click **Update Map File**.

You are prompted to restart the BMC Impact Event Adapters.

6 Click **Yes** to restart the BMC Impact Event Adapters.

The modifications made to the event classes in the **mcsnmptrapd.map** file are reflected only when you restart the BMC Impact Event Adapters.

NOTE

When you modify the variables for an event class, an asterisk (*) is displayed next to that event class in the tree structure view. The asterisk indicates that the **mcsnmptrapd.map** file has been modified on your local computer.

Unpublishing MIB files

Unpublishing MIB files is the process of removing MIB file entries from the **mcsnmptrapd.map** file.

To unpublish MIB files

1 Click **Unpublish MIBs** in the navigation bar.

2 Select the MIB file you want to unpublish. The MIB files are listed in an alphabetical order.

Press the **Shift** or **Ctrl** key to select multiple files.

The **Unpublish** button is enabled only when you select at least one MIB file.

WARNING

You cannot unpublish a MIB file if other MIB files in the **Net-SNMP** directory are dependent on it. To unpublish the parent MIB file, you have to first unpublish all the dependent MIB files. However, if you select the parent MIB file and all its dependent MIB files, you can unpublish them simultaneously.

3 To unpublish the MIB files, click **Unpublish**. You are prompted to restart the BMC Impact Event Adapters.

4 Click **Yes** to restart the BMC Impact Event Adapters.

The BMC Impact Event Adapters will stop receiving events from the devices whose MIB files have been unpublished only after you restart the BMC Impact Event Adapters. Click **No** if you do not want to restart the BMC Impact Event Adapters at this point.

You can view the status of the unpublish MIBs process in the Unpublishing Messages area.

For more information about the steps involved in the unpublish MIBs process, see Appendix A, “Backend processes for publishing and unpublishing MIB files.”

Installing the BAROC files in the Knowledge Base of a cell

A cell can start receiving events from the devices whose MIB files you have published only after you install the BAROC files created by the **mib2map.pl** utility in the Knowledge Base of the cell.

You have to install the **mcsnmptrapdmib.baroc** and **mcsnmptrapdmibe.baroc** files created by the **mib2map.pl** utility in the MCELL_HOME**etc*****_CellName_****kb****classes** directory on each cell that can receive corresponding events.

To install the BAROC files in the Knowledge Base of a cell

1. Navigate to the ***MCELL_HOME*\bin** directory on the BMC Impact Event Adapters.
2. Select the **mcsnmptrapdmib.baroc** and **mcsnmptrapdmibe.baroc** files, right-click and select **Copy**.
3. Install the files in the MCELL_HOME**etc*****_CellName_****kb****classes** directory on the cell that can receive corresponding events.
4. Ensure that the read-write permission has been set for the **.load** file.
5. Update the **.load** file to include the **mcsnmptrapdmib** and **mcsnmptrapdmibe** files, if they are not already listed.

   Though the entries do not have to be listed consecutively in the **.load** file, it is important that **mcsnmptrapdmibe** is listed before **mcsnmptrapdmib**.
6. Compile the **mcsnmptrapdmib.baroc** and **mcsnmptrapdmibe.baroc** files.
7. Restart the cell.
8. Repeat step 3 through step 7 for each cell that can receive corresponding events.

Chapter 6

Using the BMC Impact Event Log Adapter for Windows

The BMC Impact Event Log Adapter for Windows **(**BMC IELA**)** is installed separately from the other BMC Impact Event Adapters and does not use the engine manager process, the **mcxa.conf** file, or the **mceventlog.map** file. Instead, it runs as a service. BMC Software recommends that you use this Adapter process for monitoring the Windows Event Log instead of the Perl Event Log Adapter for Windows.

Overview 79
Event class definitions 79
Testing the BMC Impact Event Log Adapter for Windows 80

Overview

The BMC Impact Event Log Adapter for Windows runs as a service. It reads events generated on Microsoft Windows operating systems, formats them into the BAROC language, and forwards the events to the cell.

Event gathering occurs from three basic event logs maintained by the Microsoft Event Log service: system, application, and security. The Windows 2003 platform provides these additional event logs—DNS Server, File Replication Service, and Directory Service. The Adapter can automatically discover these event logs, so if new event logs are defined, basic events are produced.

Event class definitions

Adapter BAROC classes are defined in the **mcxa.baroc** file. This file also contains classes for standard events sent by the `msend` CLI. This file should be copied into the destination cell when the BMC IELA is installed.

The BMC IELA generates events in the following classes:

- WIN_EVENTLOG
- WIN_EL_APPLICATION
- WIN_EL_SECURITY
- WIN_EL_SYSTEM
- WIN_EL_DNS_SERVER
- WIN_EL_DIRECTORY_SERVICE
- WIN_EL_FILE_REPLICATION_SERVICE

User name, domain, and type information are automatically looked up in the Windows registry using the user SID field in the event log records.

WIN_EVENTLOG is the basic class. If the read event does not belong to any of the other classes listed above, it belongs to WIN_EVENTLOG.

Testing the BMC Impact Event Log Adapter for Windows

Confirm that the Adapter is functioning properly by using the Windows Performance monitor to generate and send test events to the cell. Use the console to view the test events.

To use the Performance Monitor generate test events

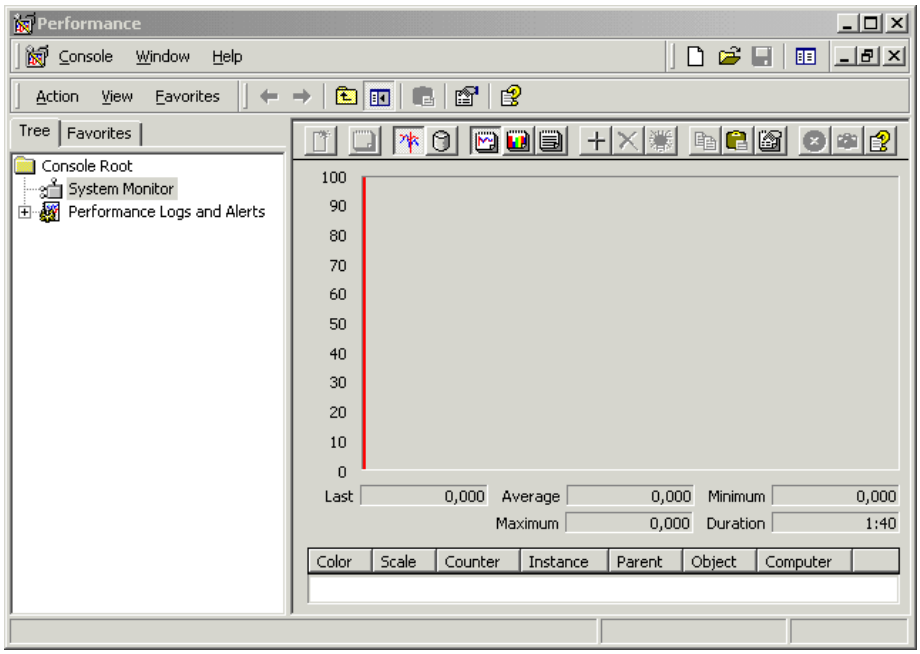
1 Choose the **Start => Programs => Administrative Tools (Common) => Performance**.

The Performance window is displayed in the Chart display mode, as shown in Figure 13 on page 81.

NOTE

The method for starting the Performance Monitor may vary depending on the version of Microsoft Windows that you are using.

Figure 13 Performance Monitor in Chart mode



2 Select the **Performance Logs and Alerts** folder in the left pane.

The Alerts icon and available logs icons appear in the right pane.

3 Right-click the **Alerts** icon and select **New Alerts Setting**.

4 In the New Alerts Settings dialog box, enter the name of the new alert and click **OK**.

The setting window for the new alert is displayed.

5 On the **General** tab, click **Add**.

6 Select Processor from the Performance objects option.

7 Enable the Select counters from list option.

8 Select **% Processor Time** under Select counters from list.

9 Enable the Select instances from list option.

10 Click **Add**.

The alert is added to the Performance Monitor and the Event Log.

11 Click **Close**.

The **Alert** window displays the values you selected.

12 In the Alert when the value is drop-down list, select Over.

13 In the **Limit** text field, enter **10**.

You may have to lower this value if 10% does not generate any test events on your system.

14 On the **Action** tab, enable the **Log an entry in the application eventlog** option.

15 Click **OK**.

You use the Performance Monitor to create test events, perform a few small operations, such as opening an application, maximizing and minimizing an application window, or moving the mouse vigorously.

Allow the Performance Monitor to generate alerts during the test. Events display in the Event Viewer window.

To view events in the BMC Impact Manager console

1 Start the BMC Impact Manager Console and connect to the cell.

2 Select the cell and click the **View Event List** button.

The Event Display window displays the test events generated by the BMC Impact Manager Event Log Adapter.

Chapter 7

IP Adapters

This chapter presents the following topics:

Overview 83
IP Adapter architecture 84
IP Adapters configuration 84
IP Adapter types 85
TCP Client Adapter 85
TCP Server Adapter 87
Telnet Adapter 90
UDP Client Adapter 94
UDP Server Adapter 96

Overview

The IP Adapters use the various protocols of the IP protocol suite to establish connections with programs from which you want to generate events.

With the IP Adapters, you are able to specify the connection method to the data source and the type of parsing that you want performed on the collected data.

Data that is collected using an IP connection is parsed into a Perl hash (name-value pair data tokens) by a parser that you specify. The parsed data is passed to a mapping function, which converts the data tokens into events. The events are sent to msend, which passes them to the cell.

For information about how and when an Adapter collects data, see "IP Adapter architecture" on page 84.

The IP Adapters are bundled with three pluggable parsers. You specify which parser you want to use as a parameter in the Adapter configuration. For more information about parsers, see Chapter 8, "BMC Impact Event Adapters parsers."

IP Adapter architecture

The Adapter architecture is a loop that is a central waiting state for a program.

When an Adapter is started, it registers to the loop. The loop checks to see if any file descriptors are active. When a descriptor becomes active, the Adapter receives the new data.

IP Adapters configuration

The Adapter configuration is contained in a list of parameters that are recorded in the Adapter definition in the **mcxa.conf** file. You configure an Adapter by specifying values for that Adapter's parameters. For information about configuring an Adapter definition, see "Adapter configuration tasks" on page 22.

Each Adapter has unique parameters that apply only to that instance of the parameter. These parameters are listed under the description for each parameter type in "IP Adapter types" on page 85.

In addition, global and common parameters also determine the configuration of an Adapter parameter instance (where instance-specific parameters do not override them).

The global parameters that affect IP Adapters are:

- `EngineMgrName`
- `PollInterval`
- `TraceFile`
- `TraceLevel`
- `TraceSizeCount`
- `TraceSizeMax`
- `TraceSizePeriod`

The common parameters that affect IP Adapters are:

- `MapFile`
- `MapFunctionsName`
- `MapUseDefaultClass`
- `ServerName`

IP Adapter types

The BMC Impact Event Adapters include a number of different IP Adapters. With the variety of different Adapters that are available, you can select the means for collecting event data that is most appropriate to your requirements.

The IP Adapters are:

- "TCP Client Adapter" on page 85
- "TCP Server Adapter" on page 87
- "Telnet Adapter" on page 90
- "UDP Client Adapter" on page 94
- "UDP Server Adapter" on page 96

The parameter descriptions include tables that describe the specific parameters of each Adapter type. Specific parameters determine how a single instance of an Adapter behaves. However, Adapter behavior is also determined by global and common parameters. The settings of global and common parameters affect multiple instances of various types of Adapters.

TCP Client Adapter

The TCP Client Adapter connects to a TCP Server to receive a data stream. The Adapter passes the data stream to a parser. The parser parses the data stream into data tokens, which the mapping function converts into events.

Table 16 on page 85 describes the TCP Client Adapter parameters.

Table 16 TCP Client Adapter configuration parameters (part 1 of 3)

Parameter	Description
instanceName	name of the Adapter instance The instance name is the first entry in the Adapter configuration and must be enclosed in square brackets. required **Note**: To run more than one instance of the Adapter, specify a unique name and configuration for each Adapter instance.
`ConnectTimeout`	maximum time, in seconds, to wait until a connection attempt is considered unsuccessful Default: **10**

Table 16 TCP Client Adapter configuration parameters (part 2 of 3)

Parameter	Description
`Engine`	type of Adapter in use required Valid value: **MA::Adapter::TcpClient**
`LocalHost`	local IP address on a computer with multiple network interfaces to which the TCP Client Adapter binds Specify a value for this parameter only when the Adapter is on a computer that has multiple network interfaces. By default, the Adapter will bind to the default interface. Valid values: host name or IP address **Note**: Do not specify **localhost**, **127.0.0.1**, or **INADDR_ANY** as a value for this parameter.
`LocalPort`	local port number to which the TCP Client Adapter binds Default: **ANY**
`Mapfile`	name of the MAP file used to generate events from the tokenized data required

`Parser`	parser instance to use to parse and tokenize the client input stream
	
	required
	
	You must code the name of another section in the **mcxa.conf** file that defines a parser with a unique name. It is not valid to code the name of a parser type as would be used in the `Engine` parameter.
	
	Example:
	
	`Parser = ParseSyslog`
	
	where `ParseSyslog` is the unique name for a parser definition, such as:
	
	`[ParseSyslog]`
	`Engine = MA::Parser::FixedWidth`
	
	In the Adapter definition, do not code:
	`Parser = MA::Parser::FixedWidth`

`RemoteHost`	hostname or IP address of the remote computer to which the TCP Client Adapter is connecting required

Table 16 TCP Client Adapter configuration parameters (part 3 of 3)

Parameter	Description
RemotePort	port on the remote computer to which the TCP Client Adapter is connecting required
RestartInterval	interval, in seconds, at which: ■ the TCP Client Adapter re-attempts to bind to the local port if that port is busy when the Adapter makes its first bind attempt ■ the TCP Client Adapter sends a packet if the connection that the Adapter uses is functioning **Note**: If the current connection fails or the remote server stops responding, this parameter value is also used to schedule additional connection attempts by the Adapter. Default: **60**

Figure 14 shows a sample TCP Client Adapter configuration, as it would appear in the **mcxa.conf** file.

Figure 14 Sample TCP Client Adapter configuration

```
[TcpClient] DISABLE
Engine = MA::Adapter::TCPClient
Parser = SeparatorParser
MapFile = mctcpclt.map
RemoteHost = localhost
RemotePort = 13       # date/time service
```

Diagnostics

All Adapter errors are logged as events. Optionally, you can enable tracing by setting the tracing parameters in the global section of the **mcxa.conf** file.

The TCP Client Adapter can generate an ERR_RECV error, which indicates that a problem occurred when receiving data from the server. For more information, see the event that is created for this error.

TCP Server Adapter

The TCP Server Adapter receives connections from one or more TCP clients. These clients send data streams to the Adapter. The Adapter passes the data stream to a parser. The parser parses the data stream into data tokens, which the mapping function converts into events.

NOTE

You cannot use Telnet as a means for communicating with the TCP Server Adapter. The TCP Server Adapter does not recognize incoming Telnet commands as meaningful binary control codes. Instead, the Adapter interprets Telnet information as raw data.

Table 17 on page 88 describes the TCP Client Adapter parameters.

Table 17 TCP Server Adapter configuration parameters (part 1 of 2)

Parameter	Description
instanceName	name of the Adapter instance The instance name is the first entry in the Adapter configuration and must be enclosed in square brackets. required **Note**: To run more than one instance of the Adapter, specify a unique name and configuration for each Adapter instance.
`Engine`	type of Adapter in use required Valid value: **MA::Adapter::TcpServer**
`LocalHost`	local IP address on a computer with multiple network interfaces to which the TCP Server Adapter binds Specify a value for this parameter only when the Adapter is on a computer that has multiple network interfaces. By default, the Adapter will bind to the default interface. Valid values: host name or IP address **Note**: Do not specify **localhost**, **127.0.0.1**, or **INADDR_ANY** as a value for this parameter.
`LocalPort`	local port number on which the TCP Server Adapter listens for connections Default: **1999**
`Mapfile`	name of the MAP file used to generate events from the tokenized data required
`MaxClients`	maximum number of clients allowed to connect simultaneously to the TCP Server Adapter Default: **10**

Table 17 TCP Server Adapter configuration parameters (part 2 of 2)

Parameter	Description
Parser	parser instance that should be used to parse and tokenize the client input stream You must code the name of another section in the **mcxa.conf** file that defines a parser with a unique name. It is not valid to code the name of a parser type as would be used in the Engine parameter. **Example**: Parser = ParseSyslog where ParseSyslog is the unique name for a parser definition, such as: [ParseSyslog] Engine = MA::Parser::FixedWidth In the Adapter definition, do not code: Parser = MA::Parser::FixedWidth
RestartInterval	interval, in seconds, at which the TCP Server Adapter re-attempts to bind to the local port if that port is busy when the Adapter makes its first bind attempt Default: **30**

Figure 15 shows a sample TCP Server Adapter configuration, as it would appear in the **mcxa.conf** file.

Figure 15 Sample TCP Server Adapter Configuration

```
[MyTcpServer]
Engine = MA::Adapter::TcpServer
MaxClients = 10
RestartInterval = 30
MapFile = mctcpsrv.map
LocalHost = 127.0.0.5
LocalPort = 1999
Parser = MyFixedWidthParser
```

Diagnostics

All Adapter errors are logged as events. Optionally, you can enable tracing by setting the tracing parameters in the global section of the **mcxa.conf** file. For more information, see “About the mcxa.conf file” on page 26.

The TCP Server Adapter can generate an ERR_RECV error, which indicates that a problem occurred when receiving data from the server. For more information, see the event that is created for this error.

Telnet Adapter

The Telnet Adapter is a Telnet client that can connect to a Telnet Server to receive a data stream. The Adapter passes the data stream to a parser. The parser parses the data stream into data tokens, which the mapping function converts into events.

Use the Telnet Adapter in the following situations:

- interactivity is required (the Adapter logs on, submits a command, and receives output)
- the process that is being monitored by the Adapter is not meant to stop (such as log file monitoring)

The Telnet Adapter operates on the following principles:

- The Telnet Adapter uses the “interpret as command” functions DO, DONT, WILL, and WONT.
- The Telnet Adapter uses “suppress go ahead” (SGA) to decrease negotiation time.
- To eliminate looping problems with SGA, each command is accepted or rejected only once and the reply is consistent every time.
- The current Telnet Adapter supports a minimal set of Telnet Negotiation commands. Almost every proposal (DO) that the Telnet Adapter receives from its peer is replied to with a negation (WONT).

Table 18 on page 90 describes the Telnet Adapter parameters.

Table 18 Telnet Adapter configuration parameters (part 1 of 4)

Parameter	Description
instanceName	name of the Adapter instance The instance name is the first entry in the Adapter configuration and must be enclosed in square brackets. required **Note**: To run more than one instance of the Adapter, specify a unique name and configuration for each Adapter instance.
`ConnectTimeout`	maximum time, in seconds, to wait until a connection attempt is considered unsuccessful Default: **10**

Table 18 Telnet Adapter configuration parameters (part 2 of 4)

Parameter	Description
Engine	type of Adapter in use required Valid value: **MA::Adapter::Telnet**
IdleTimeout	maximum time, in seconds, after which new data is expected to have arrived. After this time, the connection is considered unsuccessful. Default: **15** seconds
LocalHost	local IP address on a computer with multiple network interfaces to which the Telnet Adapter binds Specify a value for this parameter only when the Adapter is on a computer that has multiple network interfaces. By default, the Adapter will bind to the default interface. Valid values: host name or IP address **Note**: Do not specify **localhost**, **127.0.0.1**, or **INADDR_ANY** as a value for this parameter.
LocalPort	local port number to which the Telnet Client binds Default: **ANY**
LoginPrompt	regular expression of the login prompt required Default: "**login[:]*$ | username[:]*$**"
Mapfile	name of the MAP file used to generate events from the tokenized data required

Parser	parser instance that should be used to parse and tokenize the client input stream
	
	required
	
	Example:
	
	`Parser = ParseSyslog`
	
	where ParseSyslog is the unique name for a parser definition, such as:
	
	`[ParseSyslog]`
	`Engine = MA::Parser::FixedWidth`
	
	In the Adapter definition, do not code:
	
	`Parser = MA::Parser::FixedWidth`

Table 18 Telnet Adapter configuration parameters (part 3 of 4)

Parameter	Description
PasswordPrompt	regular expression that provides valid alternative password prompt text required Default: "**password[:]*$**"
Prompt	list of alternate prompt characters required Default: "**[\$>#%]**"
script	Telnet login script required Default: **telnet.conf**
RemoteHost	hostname or IP address of the remote computer to which the Telnet Adapter is connecting required
RemotePort	port on the remote computer to which the Telnet Adapter is connecting required
RestartInterval	interval at which the Telnet Adapter re-attempts to bind to the local port if that port is busy when the Adapter makes its first bind attempt This interval also can be used for any other rescheduling that happens within the Adapter (such as rescheduling timeout or termination). Default: **60**
RestartWhenDone	reestablishes a connection when it fails or is broken due to a hang-up Valid values: ■ **0** – no reconnect ■ **1** – reconnect Default: **0**
TelnetAuthTimeout	maximum time, in seconds, to wait for the completion of the authentication phase required
TelnetCommand	command to execute after logging onto the remote computer required

Table 18 Telnet Adapter configuration parameters (part 4 of 4)

Parameter	Description
TelnetPassword	password used to connect to the remote computer required
TelnetUsername	user name used to connect to the remote computer required

Figure 16 shows a sample Telnet Adapter configuration, as it would appear in the **mcxa.conf** file.

Figure 16 Sample Telnet Adapter configuration

```
[MyTelnet] #name of Adapter instance
Engine = MA::Adapter::Telnet
Parser = SeparatorParser
# Use the parser you want for the output
MapFile = mctelnet.map
RestartInterval = 60 # Interval used when an error occurs
# Or when Adapter restarts
IdleTimeout = 15 # Close the connection when no data has
# been received for 10 seconds (implicit restart)
ConnectTimeout = 10 # Restart if the connection takes longer
# Than 10 seconds to complete
RemoteHost = localhost   # Which host to connect to
RemotePort = 23 # Which port (23 = telnetd port)
TelnetAuthTimeout = 15
TelnetUsername = myusername  # Username for login
TelnetPassword = mypasswd    # Password for login
TelnetCommand = "ls -l"     # Command to execute after login
```

Diagnostics

All Telnet Adapter errors are logged as events. Optionally, you can enable tracing by setting the tracing parameters in the global section of the **mcxa.conf** file.

The Telnet Adapter can generate an ERR_RECV error, which indicates that a problem occurred when receiving data from the server. For more information, see the event that is created for this error.

NOTE

If you suspect problems with the Telnet Negotiation commands, monitor the Telnet option negotiation process by enabling Trace level 6. For details about tracing, see "Enabling tracing" on page 67.

UDP Client Adapter

The UDP Client Adapter connects to a UDP Server to receive a data stream. The Adapter passes the data stream to a parser. The parser parses the data stream into data tokens, which the mapping function converts into events.

Table 19 describes the UDP Client Adapter parameters.

Table 19 UDP Client Adapter configuration parameters (part 1 of 2)

Parameter	Description
instanceName	name of the Adapter instance The instance name is the first entry in the Adapter configuration and must be enclosed in square brackets. required ■ **Note**: To run more than one instance of the Adapter, specify a unique name and configuration for each Adapter instance.
`ConnectTimeout`	maximum time a connection will stay active After the period of time specified in ConnectTimeout has expired, the Adapter closes the connection. No more data will be received from that connection. If no `ConnectTimeout` value is specified or if the value specified is **0**, the timer is disabled. This is the default behavior.
`Engine`	type of Adapter in use. required Valid value: **MA::Adapter::UdpClient**
`IdleTimeout`	maximum time, in seconds, after which new data is expected to have arrived. After this time, the connection is considered unsuccessful. Default: **15** The UDP Client Adapter sends a packet at a regular interval, specified by the sum of the periods specified by the `IdleTimeout` and `RestartInterval` parameters.
`LocalHost`	local IP address on a computer with multiple network interfaces to which the UDP Client Adapter binds Specify a value for this parameter only when the Adapter is on a computer that has multiple network interfaces. By default, the Adapter will bind to the default interface. Valid values: host name or IP address **Warning**: Do not specify **localhost**, **127.0.0.1**, or **INADDR_ANY** as a value for this parameter.

Table 19 UDP Client Adapter configuration parameters (part 2 of 2)

Parameter	Description
LocalPort	local port number to which the UDP Client Adapter binds Default: **ANY**
Mapfile	name of the MAP file used to generate events from the tokenized data required

Parser	parser instance that should be used to parse and tokenize the client input stream
	
	required
	
	Example:
	
	`Parser = ParseSyslog`
	
	where `ParseSyslog` is the unique name for a parser definition, such as:
	
	`[ParseSyslog]`
	`Engine = MA::Parser::FixedWidth`
	
	In the Adapter definition, do not code:
	
	`Parser = MA::Parser::FixedWidth`

RemoteHost	hostname or IP address of the remote computer to which the UDP Client Adapter is connecting
RemotePort	port on the remote computer to which the UDP Client Adapter is connecting
RestartInterval	interval, in seconds, at which ■ the UDP Client Adapter re-attempts to bind to the local port if that port is busy when the Adapter makes its first bind attempt ■ the UDP Client Adapter sends a packet at the rate prescribed by this parameter plus the interval prescribed in the `IdleTimeout` parameter, if the connection that the Adapter uses is functioning Default: **60** The `RestartInterval` parameter value is also used to schedule additional connection attempts by the Adapter when any of the following occur: ■ the current connection fails ■ the remote server hangs ■ the amount of time specified in the `IdleTimeout` parameter expires
SendOnConnect	string that is sent after a connection is established

Figure 17 on page 96 shows a sample UDP Client Adapter configuration, as it would appear in the **mcxa.conf** file.

Figure 17 Sample UDP client Adapter configuration

```
[MyUDPClient]
Engine = MA::Adapter::UdpClient
MapFile = mcudpclt.map
RemoteHost = test.bmc.com
RemotePort = 13
Parser = SeparatorParser
```

Diagnostics

All Adapter errors are logged as events. Optionally, you can enable tracing by setting the tracing parameters in the global section of the **mcxa.conf** file. For more information, see "Enabling tracing" on page 67.

The UDP Client Adapter can generate an `ERR_UDP_INIT` error, which indicates that a problem occurred when receiving data from the server. For more information, see the event that is created for this error.

UDP Server Adapter

The UDP Server Adapter receives connections from one or more UDP clients. The Adapter passes the data stream to a parser. The parser parses the data stream into data tokens, which the mapping function converts into events.

Table 20 describes the UDP Server Adapter parameters.

Table 20 UDP Server Adapter configuration parameters (part 1 of 2)

Parameter	Description
instanceName	name of the Adapter instance The instance name is the first entry in the Adapter configuration and must be enclosed in square brackets. required **Note**: To run more than one instance of the Adapter, specify a unique name and configuration for each Adapter instance.
`Engine`	type of Adapter in use. required Valid value: **MA::Adapter::UdpServer**

Table 20 UDP Server Adapter configuration parameters (part 2 of 2)

Parameter	Description
`IdleTimeout`	maximum time, in seconds, after which new data is expected to have arrived. After this time, the connection is considered unsuccessful. Default: **15**
`LocalHost`	local IP address on a computer with multiple network interfaces to which the UDP Client Adapter binds Specify a value for this parameter only when the Adapter is on a computer that has multiple network interfaces. By default, the Adapter will bind to the default interface. **Note**: Do not specify **localhost**, **127.0.0.1**, or **INADDR_ANY** as a value for this parameter. Valid values: host name or IP address
`LocalPort`	local port number on which the UDP Server Adapter listens for connections Default: **2000**
`Mapfile`	name of the MAP file used to generate events from the tokenized data required

`Parser`	parser instance that should be used to parse and tokenize the client input stream
	
	required
	
	Example:
	
	`Parser = ParseSyslog`
	
	where `ParseSyslog` is the unique name for a parser definition, such as:
	
	`[ParseSyslog]`
	`Engine = MA::Parser::FixedWidth`
	
	In the Adapter definition, do not code:
	
	`Parser = MA::Parser::FixedWidth`

`RestartInterval`	interval, in seconds, at which the UDP Server Adapter re-attempts to bind to the local port if that port is busy when the Adapter makes its first bind attempt Default: **60**

Figure 18 shows a sample UDP Server Adapter configuration, as it would appear in the **mcxa.conf** file.

Figure 18 Sample UDP Server Adapter configuration

```
[MyUDPServer]
Engine = MA::Adapter::UdpServer
MapFile = mcudpsrv.map
LocalPort = 1997
Parser = SeparatorParser
```

Diagnostics

All Adapter errors are logged as events. Optionally, you can enable tracing by setting the tracing parameters in the global section of the **mcxa.conf** file. For more information, see "Enabling tracing" on page 67.

The UDP Server Adapter can generate the following errors:

- `ERR_RECV`—indicates that a problem occurred when receiving data from the server
- `ERR_UDP_INIT`—indicates that a problem occurred when starting the UDP connection

For more information, see the event that is created for this error.

Chapter 8

BMC Impact Event Adapters parsers

This appendix presents the following topics:

What a parser does 99
LogFile, Perl EventLog for Windows, and SNMP Adapter parsers 99
IP Adapter parsers 101
Specifying a CSV parser 101
Specifying a Fixed-width (column) parser 103
Specifying Record and Field separated parser 107

What a parser does

A parser formats the data that is collected by one of the BMC Impact Event Adapters, converts it into tokens (fields), and passes these tokens to the mapping process.

The SNMP Adapter, the BMC LogFile Adapter, and the Perl Windows EventLog Adapter each use a dedicated parser engine. This engine cannot be replaced or modified. You do not need to specify the parser in the Adapter definition.

The BMC Impact Event IP Adapters can use any of three pluggable parsers, which each parse the data in a different format. For each IP Adapter, you must specify a parser in the Adapter definition in the **mcxa.conf** file. The IP Adapter requests the parser as an object.

LogFile, Perl EventLog for Windows, and SNMP Adapter parsers

The LogFile Adapter, Perl EventLog for Windows Adapter, and SNMP Adapters include dedicated internal parsing functions.

Events in a log file are determined either by a record separator or by a Perl regular expression.

The most common event separator is the `newline` (`\n`). The parameter `LogRecordSeparator` contains the record separator of the events. `LogRecordSeparator` recognizes Perl escape characters, such as `\n`, and generally can contain several characters or even a regular expression. For example, setting `LogRecordSeparator="\n\n"` results in events being delimited by two consecutive new line characters that results in a single empty line. By default, setting `LogRecordSeparator = "\n"` creates events that are line-based. In other cases, the parsing will be multi-lined.

The input variable `$complete` is set to the string between the two consecutive `LogRecordSeparator` entries.

According to the value of the parameter `LogRegExpr`, the following instances are considered:

- `LogRegExpr` is empty. `$complete` is split into substrings, delimited by the `LogFieldSeparator` variable. These substrings are stored in `$varlog<N>` variables (`N>=0`). For example, if

  ```
  $complete contains "a;b;c;d", and
  $LogFieldSeparator contains ";"
  ```

  then

  ```
  $varlog0="a"
  $varlog1="b"
  ...
  ```

- `LogRegExpr` is not empty. The regular expression `LogRegExpr` is applied to `$complete`. Subpatterns of `$complete` are stored in `$varlog<N>` variables (`N>=1`). For example,

  ```
  $complete contains "12345 abcde", and
  $logRegExpr contains "(\d+)\s*(\w+)"
  ```

  then

  ```
  $varlog1=12345
  $varlog2="abcde"
  ```

- `LogRecordSeparator` is empty and `LogRegExpr` is not empty. In this case, the regular expression contained in `LogRegExpr` will be applied to the log file from the point at which the previous successful matching stopped. The input variable `$complete` will be set to the whole matching of the regular expression. `LogVarPrefix` contains the names of variables containing subpatterns. By default,

  ```
  LogVarPrefix="varlog"
  ```

  If both variables are empty, the behavior of the Adapter is not determined.

IP Adapter parsers

Unlike the Logfile Adapter, the SNMP Adapter, and the Perl EventLog for Windows Adapter, which have their own dedicated parsers, the IP Adapters can use any one of the following parsers that are included with the BMC Impact Event Adapters.

- "Specifying a CSV parser"
- "Specifying a Fixed-width (column) parser" on page 103
- "Specifying Record and Field separated parser" on page 107

Specifying a CSV parser

The CSV parser, **MA::Parser::CSV**, is a pluggable parser that parses a datastream into records (newline), in which each record is a comma-separated value list. The parser returns each value as a token to the mapper.

To specify the CSV parser

1. Using a text editor, open the **mcxa.conf** file.
2. In the section of the file that describes the Adapter that you want to use with the CSV parser, add the following line:

   ```
   Parser = parsername
   ```

3. In the parser section of the **mcxa.conf** file, define parser *parsername*.

   Table 21 on page 102 describes the parameters that you can modify in the parser definition.

Table 21 CSV parser parameters

Parameter	Description
FieldSeparator	a set of characters that will be used as a delimiter Optional. Default: comma (**,**)
KeepEmptyRecords	indicates whether empty records are formatted as tokens and passed to the mapping function required Valid values: ■ **1** – tokenize empty records ■ **0** – discard empty records (Default)
RecordNegFilter	a regular expression that acts as a criterion to which each parsed record is compared If the criterion is included in the record, the record is discarded. Optional.
RecordPosFilter	a regular expression that acts as a criterion to which each parsed record is compared If the string is included in the record, the record is formatted as a token and passed to the mapping function. Optional.
RecordSeparator	regular expression used to break a datastream into separate records The resulting record is returned in the $data variable, which can then be used in the MAP file to refer to the entire record as one field. The $data variable is provided automatically. You cannot rename it. Optional. Default: **\r?\n** **Note**: The input variable $data is set to the string between two consecutive RecordSeparator entries. It performs the function that $complete performs in the LogFile, Perl EventLog for Windows, and SNMP Adapter mappings.
VarPrefix	variable prefix used by the mapper Each token will be returned as **prefix0** ... **prefix*n***. Optional. Default: **varlog** Default variables sent to the mapper are named **varlog0, varlog1, ...**

Figure 19 shows a sample parser definition for a parser named **MyParser**.

Figure 19 Sample CSV parser specification example

```
[MyParser]
Engine = MA::Parser::CSV
FieldSeparator = ","  # Default
KeepEmptyRecords= 0    # Discard Empty Records
RecordNegFilter = "(test|dummy) event"
RecordPosFilter = "(ERROR|WARNING)"
VarPrefix = "token" # Return token0 ... tokenn
```

4 Save and close the **mcxa.conf** file.

Specifying a Fixed-width (column) parser

The fixed-width parser, **MA::Parser::FixedWidth**, is a pluggable parser that parses a datastream into records. From each record, a field is extracted based on a fixed-width format.

To specify the fixed-width parser

1 Using a text editor, open the **mcxa.conf** file.

2 In the section of the file that describes the Adapter that you want to use with the fixed-width parser, add the following line:

```
Parser = parsername
```

3 In the parser section of the **mcxa.conf** file, define parser *parsername*.

Table 22 on page 104 describes the parameters that you can modify in the parser definition.

Table 22 Fixed-Width parser parameters (part 1 of 2)

Parameter	Description
`FieldDefinitions`	a numerical pair that specifies which data are used in a field The first number in the pair specifies from which character in the string to begin collecting data. The second number specifies how many consecutive characters are to be included in the field. For more information, see "Defining Fields" on page 105. required
`KeepEmptyRecords`	indicates whether empty records are formatted as tokens and passed to the mapping function required Valid Values: ■ **1** – tokenize empty records ■ **0** – discard empty records (Default)
`RecordNegFilter`	a regular expression that acts as a criterion to which each parsed record is compared If the criterion is included in the record, the record is discarded. Optional.
`RecordPosFilter`	a regular expression that acts as a criterion to which each parsed record is compared If the string is included in the record, the record is formatted as a token and passed to the mapping function. Optional.
`RecordSeparator`	regular expression used to break a datastream into separate records The resulting record is returned in the `$data` variable, which can then be used in the MAP file to refer to the entire record as one field. The `$data` variable is provided automatically. You cannot rename it. Optional. Default: **\r?\n** **Note**: The input variable `$data` is set to the string between two consecutive `RecordSeparator` entries. It performs the function that `$complete` performs in the LogFile, Perl EventLog for Windows, and SNMP Adapter mappings.

Table 22 Fixed-Width parser parameters (part 2 of 2)

Parameter	Description
TrimLeadingWS	removes leading white space, if any, from fields required Valid Values: ■ **1** – removes white space (Default) ■ **0** – does not remove white space
TrimTrailingWS	removes trailing white space, if any, from fields required Valid Values: ■ **1** – removes white space (Default) ■ **0** – does not remove white space
VarPrefix	variable prefix used by the mapper Each token will be returned as **prefix0** ... **prefix*n***. Optional. Default: **varlog** Default variables sent to the mapper are named **varlog0, varlog1, ...**

Figure 20 shows a sample parser definition for a parser named **MyParser**.

Figure 20 Fixed-width parser specification example

```
[MyParser]
Engine = MA::Parser::FixedWidth
RecordSeparator = "\r?\n"
FieldDefinitions = "0:10,5:10,15:20"
KeepEmptyRecords = 0
TrimLeadingWS = 1
TrimTrailingWS = 1
VarPrefix = "token" # Return token0 ... tokenn
```

4 Save and close the **mcxa.conf** file.

Defining Fields

The contents of each field are determined by a pair of numbers. Each record from which the field is extracted is a string of data. The first field number indicates the starting position in the record at which the string field begins. The second number indicates the length of the string.

Defining a Single Field

The first character of the record is zero (**0**). A starting number of **5** indicates that the string begins with the sixth character in the record string.

EXAMPLE

In this example, the string is:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

If the field specification is **<0,3>**, then the string that composes the field is **ABC**. The **A** is at position **0**, and then starting from position **0**, three characters are selected to comprise the field.

If the field specification is **<11,2>**, then the string that composes the field is **LM** (remember that the first character in the record is at position **0**, not **1**). The **L** is at position **11**, and then starting from position **11**, two characters are selected to comprise the field.

Either number in the field specification can include a negative number. The field length position can include an asterisk wildcard (*****).

- When the starting point (the first number) is negative, the starting position is determined from the end of the record.

  EXAMPLE

  In this example, the string is:

  ABCDEFGHIJKLMNOPQRSTUVWXYZ

  If the field specification is **<-8,3>**, then the string that composes the field is **STU**. The **S** is at position **-8**, and then starting from position **-8**, three characters are selected to comprise the field.

  Alternately, you can express the same string in positive numbers: **<18,3>.**

- When the field length (the second number) is negative, the end point of the string is calculated by counting back that number of characters from the end of the string and including everything in between the start point and up to that end point.

  EXAMPLE

  In this example, the string is:

  ABCDEFGHIJKLMNOPQRSTUVWXYZ

  If the field specification is **<8,-3>**, then the string that comprises the field is **IJKLMNOPQRSTUVW**. The string starts at **I** (position **8**) and includes all but the last three characters in the record string.

- Both the starting point and the length can be negative values.

EXAMPLE

In this example, the string is:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

If the field specification is **<-20,-4>**, then the string that comprises the field is **GHIJKLMNOPQRSTUV**. The string starts at **G** (position **-20**) and includes all but the last four characters in the record string.

- When the length is an asterisk (*****), all characters from the starting point to the end of the record string comprise the field string. Use the asterisk when you do not know the record field length but want to select all of it from the starting point that you specify.

EXAMPLE

In this example, the string is:

ABCDEFGHIJKLMNOPQRSTUVWXYZABC

If the field specification is **<15,*>**, then the string that comprises the field is **PQRSTUVWXYZABC**. The string starts at **P** (position **15**) and includes all remaining characters in the record string.

Defining Multiple Fields

Each record can contain multiple fields. A prototype for their order is

```
<START1>:<LENGTH1>[, <START2>:<LENGTH2>[, <STARTn>:<LENGTHn>[...]]]
```

Specifying Record and Field separated parser

The Record and Field Separated parser, **MA::Parser::Separator**, is a pluggable parser that parses a datastream into user-definable records. Each record is then converted into a token using a user-defineable field separator.

To specify the Record and field separated parser

1 Using a text editor, open the **mcxa.conf** file.

2 In the section of the file that describes the Adapter that you want to use with the record and field separated parser, add the following line:

```
Parser = parsername
```

3 In the parser section of the file, define parser *parsername*.

The parameters of the parser definition that you can modify are described in Table 23.

Table 23 Record and field separated parser parameters (part 1 of 2)

Parameter	Description
FieldSeparator	regular expression used to break a record into multiple fields required Default: **\s+** (separates on any instance of white space)
KeepEmptyRecords	indicates whether empty records are formatted as tokens and passed to the mapping function required Valid Values: ■ **1** – tokenize empty records ■ **0** – discard empty records (Default)
RecordNegFilter	a regular expression that acts as a criterion to which each parsed record is compared If the criterion is included in the record, the record is discarded. Optional.
RecordPosFilter	a regular expression that acts as a criterion to which each parsed record is compared If the string is included in the record, the record is formatted as a token and passed to the mapping function. Optional.

Table 23 Record and field separated parser parameters (part 2 of 2)

Parameter	Description
RecordSeparator	regular expression used to break a datastream into separate records The resulting record is returned in the $data variable, which can then be used in the MAP file to refer to the entire record as one field. The $data variable is provided automatically. You cannot rename it. Optional. Default: **\r?\n** **Note**: The input variable $data is set to the string between two consecutive RecordSeparator entries. It performs the function that $complete performs in the LogFile, Perl EventLog for Windows, and SNMP Adapter mappings.
TrimLeadingWS	trims the leading white space from the *record* before applying the RecordSeparator, which is done using the split function In this parser, TrimLeadingWS does not trim the white space from each FIELD after the RecordSeparator has been applied.
VarPrefix	variable prefix used by the mapper. Each token will be returned as **prefix0** ... **prefix*n***. Optional. Default: **varlog** Default variables sent to the mapper are named **varlog0, varlog1, ...**

Figure 21 shows a sample parser definition for a parser named **MyParser**.

Figure 21 Record and Field Separated Parser Specification Example

```
[MyParser]
Engine = MA::Parser::Separator
RecordSeparator = "\r?\n"
FieldSeparator = "\s+"
KeepEmptyRecords = 0    # Discard Empty Records
RecordNegFilter = "(test|dummy) event"
RecordPosFilter = "(ERROR|WARNING)"
VarPrefix = "token" # Return token0 ... tokenn
```

4 Save and close the **mcxa.conf** file.

Appendix A

Backend processes for publishing and unpublishing MIB files

This appendix presents the following topics:

The publish MIB files back-end process 111
The unpublish MIB files back-end process 112

The publish MIB files back-end process

The cell automates the publish MIBs process. The steps involved in the publish MIB files back-end process have been reproduced here for your reference only.

1. The new MIB files are moved from your local computer to the **Net-SNMP** directory.

2. Before overwriting a MIB file that already exists in the **Net-SNMP** directory, a backup of that file is saved in the ***MCELL_HOME*\tmp\Adapters\snmpAdapter\mibs** directory.

3. The **mib2map.pl** utility, which is located in the ***MCELL_HOME*\bin** directory, is run. It generates the following four output files:

   — **mcsnmptrapdmib.baroc**: This file contains the BAROC class definitions of the SNMP Traps found.

   — **mcsnmptrapdmibe.baroc**: This file contains the BAROC enumerations of the MIB enumerated variables that are sent by traps.

   — **mcsnmptrapdmib.dat**: This file contains information about object identifiers (OIDs), symbolic names, enumerations and indexes found in MIB traps.

   — **mcsnmptrapdmib.map**: This file converts traps into BAROC.

4. The four output files are saved in the ***MCELL_HOME*\bin** directory.

5. A backup of the previous **mcsnmptrapd.dat** file is saved in the ***MCELL_HOME*\tmp\Adapters\snmpAdapter\mib2mapOutput** directory.

6. A backup of the previous **mcsnmptrapd.map** file is saved in the ***MCELL_HOME*\tmp\Adapters\snmpAdapter\map** directory. The file is renamed to **mcsnmptrapdLastModified.map**.

NOTE

- In the **mcxa.conf** file, if you have not specified the file names for the `MapFile` and `SnmpDatFile` parameters, by default the Product Name uses the **mcsnmptrapd.map** and **mcsnmptrapd.dat** file names respectively. This document uses **mcsnmptrapd.map** and **mcsnmptrapd.dat** to refer to the files that are configured for the `MapFile` and `SnmpDatFile` parameters respectively in the **mcxa.conf** file.

- In the **mcxa.conf** file, if you have not specified the file name for the `MapFile` parameter, the Product Name renames the **mcsnmptrapd.map** file to **mcsnmptrapdLastModified.map**. Otherwise, 'mcsnmptrapd' is replaced with the file name that you have specified for the `MapFile` parameter in the **mcxa.conf** file. For example, *filename***LastModified.map**.

7. The **mcsnmptrapdmib.dat** and **mcsnmptrapdmib.map** files are renamed to **mcsnmptrapd.dat** and **mcsnmptrapd.map** respectively, and moved to the ***MCELL_HOME*\etc** directory.

8. The customizations in the previous **mcsnmptrapd.map** file are merged with the new **mcsnmptrapd.map** file.

NOTE

The Publish MIBs process preserves the entries of classes in the **mcsnmptrapd.map** file even if the corresponding MIB files do not exist in the **Net-SNMP** directory.

All the invalid MIB files in the **Net-SNMP** directory are identified and displayed in the status area. Before you manually delete an invalid file existing in the **Net-SNMP** directory, ensure that the invalid file does not support any valid MIB files.

The unpublish MIB files back-end process

The cell automates the unpublish MIB files process. The steps involved in the unpublish MIB files back-end process have been reproduced here for your reference only.

1. A backup of the **mcsnmptrapd.map** and **mcsnmptrapd.dat** files is saved in the ***MCELL_HOME*\backup\Adapters\snmpAdapter\removeMib** directory. These two files are located in the ***MCELL_HOME*\etc** directory.

2. The **mcsnmptrapd.map** template file is copied to the ***MCELL_HOME*\etc** directory.

3. A backup of the **mcsnmptrapdmib.baroc** and **mcsnmptrapdmibe.baroc** files is saved in the ***MCELL_HOME*\backup\Adapters\snmpAdapter\removeMib** directory. These two files are located in the ***MCELL_HOME*\bin** directory.

4. The following tasks are repeated for each MIB file that has to be unpublished.

   A. The modules within the MIB file that has to be unpublished are identified.

   B. The **mib2map.pl** utility, which is located in the ***MCELL_HOME*\bin** directory, is run only for the MIB file that has to be unpublished. The four output files that are generated are **mcsnmptrapdmib.baroc**, **mcsnmptrapdmibe.baroc**, **mcsnmptrapdmib.dat**, and **mcsnmptrapdmib.map**. The four output files are saved in the ***MCELL_HOME*\bin** directory.

   C. The classes created for the MIB file are extracted from the **mcsnmptrapdmib.map** file, which is located in the ***MCELL_HOME*\bin** directory. The same classes are then removed from the **mcsnmptrapd.map** file, which is located in the ***MCELL_HOME*\etc** directory.

   D. The content of the **mcsnmptrapd.dat** file, which is located in the ***MCELL_HOME*\bin** directory, is referenced and then the same content is removed from the **mcsnmptrapd.dat** file, which is located in the ***MCELL_HOME*\etc** directory.

5. The backup of the **mcsnmptrapdmib.baroc** and **mcsnmptrapdmibe.baroc** files, which are located in the ***MCELL_HOME*\backup\Adapters\snmpAdapter\removeMib** directory, is restored to the ***MCELL_HOME*\bin** directory.

6. The MIB files that are unpublished are also removed from the **Net-SNMP** directory.

Index

Symbols

$data variable 102, 104, 109

A

adapter engine 27
adapters
  BAROC classes 44
  configuration file 15
  configuration file, described 14, 15
  default configuration file 26
  default engines 27
  Engine parameter specifies type 27
  EventLog 79
  global parameters 27
  integration of user-defined 19
  mcxa.conf 26
  multiple instances of same type 15
  SNMP (trap) 16
  starting 61
  stopping 63
  tracing 67
architecture
  IP adapters 84
assignments, map files, slots 48

B

BAROC
  classes, mcxa.baroc file 44
BMC Software, contacting 2

C

classes
  map file sections 46
  map files 53
commands
  kill 63
comments, configuration file 28
common parameters 30
  defined 29
  definitions 29
conditions
  for map files 50
configuration files
  adapters 15
  comments 28
  definitions 28
  Engine parameter 19
  MAP file 44
  parameters 28
  sections 28
  structure 26
configuring
  TCP Server adapter 88
  telnet adapter 90
  UDP Client adapter 94
  UDP Server adapter 96
csv parser
  description 101
customer support 2

D

default adapters configuration file 26
DEFAULT map file sections 45, 53
default, map files 44
DEFINE, map file directive 56
definitions, configuration file 28
diagnostics
  TCP client adapter 87
  TCP Server adapter 89
  telnet adapter 93
  UDP Client adapter 96
  UDP Server adapter 98

E

Engine parameter
  adapter type, specified by 27
  configuration file 19
error events 68
event classes
  definition for adapters 14, 15
EventLog adapter 79
events
  error 68

MC_ADAPTER_ERROR 68
MC_ADAPTER_START 68
MC_ADAPTER_STOP 68

F

fixed-width parser
description 103
functions
map files, slots 49

G

global parameters 30
adapters 27
list of 30

I

INIT 52
map file sections 45
section, map file 52
INPUT_VARIABLES, map file sections 45, 46
instance control 64
IP adapters
architecture 84
instance control 64
list of types 85
overview 83
TCP client adapter description 85
TCP Server adapter description 87
telnet adapter description 90
UDP Client adapter description 94
UDP Server adapter description 96

K

kill command 63

L

LogFile Adapter, parsing 99

M

map files
Class structure 53
conditions 50
configuration file 44
contents of 15
default 44
described 14, 15
directives
DEFINE 56
REQUIRE 56
INIT 45
operators 50
personalized 45
REQUIRE directive 56
sections
CLASS 46
DEFAULT 45, 53
INIT 52
INPUT_VARIABLES 45, 46
slots
assignments 48
functions 49
structure 45
MC_ADAPTER_ERROR event 68
MC_ADAPTER_START event 68
MC_ADAPTER_STOP event. 68
mcxactrl.pl command
description 64

O

operators, map files 50
overview, IP adapters 83

P

parameter types
common 29
global 29
specific 29
parameters
common
definitions of 29
configuration file 28
definitions of common 29
destination Cell name 28
list of 30
specific
Logfile, characteristics 35
Logfile, list of 35
SNMP Trap, characteristics 40
SNMP Trap, list of 40
TraceFile 67
TraceFileCount 67
TraceLevel 67
parsers
csv parser 101
fixed-width parser 103
separator parser 107
parsing engine, described 13, 15
Perl interpreter, regular expression 15
personalized map files 45

product support 2

R

Regular expressions,Perl interpreter 15
REQUIRE directive, map files 56
REQUIRE, map file directive 56

S

sections, configuration file 28
separator parser
    description 107
slots
    assignments, map files 48
    functions, map files 49
SNMP
    (trap) adapter 16
specific parameters 30
starting
    adapters 61
starting Adapters
    using mcxactrl.pl 66
stopping adapters 63
structure, map files 45
support, customer 2

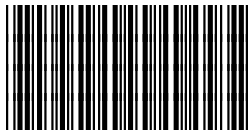
T

TCP client adapter
    description 85
    diagnostics 87
TCP Server adapter
    description 87
    diagnostics 89
TCP server Adapter
    configuration parameters 88
technical support 2
telnet adapter
    configuration parameters 90
    description 90
    diagnostics 93
TraceFile parameter 67
TraceFileCount parameter 67
TraceLevel parameter 67
tracing 67

U

UDP Client adapter
    configuration parameters 94
    description 94
    diagnostics 96
UDP Server adapter
    configuration parameters 96
    diagnostics 98
user-defined adapters 19
using mcxactrl.pl 66

Notes



106569